SPEECH RECOGNITION USING VISIBLE AND INFRARED DETECTORS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirement for the Degree of
Master of Science in Electrical Engineering

Patrick T. Marshall
GS-12, USAF

September 1992

Acknowledgments

Many thanks to my fellow Hanger Rats who I work with and who helped me make this research effort become a success. I want to thank my supervisor, Mr. Daniel Murray, who supported me throughout this effort. I want to also thank my thesis advisor, Dr. Matthew Kabrisky, who not only provided the technical expertise to keep my research going in the right direction but who also allowed flexibility to explore my ideas. Finally, I would like to give a special thanks to my wife Janet for her support and encouragement during the last six years.

Patrick T. Marshall

Table of Contents

List of Figures

List of Tables

Abstract

A system has been developed that tracks lip motion using infrared (IR) or visible detectors. The purpose of this study was to determine if the additional information obtained from the IR or visible detectors can be used to increase the recognition rate of audio Automatic Speech Recognition (ASR) systems. To accomplish this goal, several hardware analog prototypes had to be designed, built and tested. Different detectors (IR and visible) and modes of operation (active and passive) were tried before a reliable and useful signal was found. An analog-to-digital (A/D) board was then designed and built that digitized both the microphone and photo signals. Software algorithms, executed from a desktop PC, were used to interface with the A/D board, process the digitized data, and perform certain optical and audio ASR experiments. The results showed that isolated ASR audio recognition rates increased after using additional information gained from the photo speech signals. However, the results for the continuous case were inconclusive since not all of the available photo information was utilized to perform ASR experiments.

SPEECH RECOGNITION USING VISIBLE AND INFRARED DETECTORS

I. Introduction

Speech Recognition is the process that converts acoustic sound waves generated by human organs into the equivalent of a type script file. ASR technology is the human-machine interface (hardware and/or software) that accomplishes this task without any outside assistance. For decades now scientists and engineers have been trying, with some success under carefully controlled conditions, to create such a system. Most of their time and efforts have been concentrated on deriving and/or implementing special algorithms that process audio information only. However, there is additional information other than the audio signal that could be used in the recognition process. Deaf people, who are trained lip-readers, use this information by observing certain visual queues produced by the mouth and surrounding areas. This same information is potentially available to ASR systems which, if usable, will most certainly increase recognition rates.

Why is ASR technology so important? From a military standpoint it should help bridge the distance between the current generation and next generation aircraft. This is because the present human-machine interface in cockpit technology seems to be nearing its limits. Current cockpit technology frequently requires the pilot to flip some switches or turn a knob while usually monitoring a LED (light emitting diode) readout. Voice-controlled avionics could allow the pilot to command his aircraft simply by talking to it. The end result should be an increase in fighting performance to help insure that the United States Air Force maintains its air superiority.

ASR technology could also be important for some office and industrial applications. It would be much simpler to talk to a computer while data processing. However, it is the keyboard that is currently the major player for inputting data to a computer. Simply put, there is no commercially available ASR system that is fully operational at a reasonable price that could even begin to replace the computer keyboard.

1.1 Background

Basically there are two modes of operation for speech recognition: isolated and continuous. By definition, isolated speech recognition is the identification of singly spoken words. In other words, the speaker can only say one word per recognition run. Equipment using this mode of operation is commercially available and has enjoyed some success in carefully controlled applications. This is due, at least in part, to the fact that the speaker has to insert a pause between words. In low noise environments the computer can easily find the word boundaries and activate the necessary algorithms to match the entire word utterance against stored prototypes of allowed words. However, it is unnatural for humans to be forced to put pauses between words while talking.

Continuous speech recognition is the ability to recognize words in natural speech (i.e., no forced pauses are required). This mode of operation is the potentially most important one and unfortunately is also the hardest one to accomplish. The biggest problem is trying to find where words begin and end in a sentence. However, this problem may be lessened with the use of additional information in the recognition process.

An optical ASR system incorporates visual and/or infrared (IR) information to help increase recognition rates. Most reported research into optical ASR utilized a video camera to record visual data followed by the analysis of static video images to perform optical ASR [1]. The basic theory of operation is to digitize video images of the mouth at certain positions and use them as templates while performing ASR. Unfortunately these are very computationally-intensive systems. The camera alone would output video data at "5 mega-bytes per second." In addition, a data reduction of about 7500 to 1 had to be performed on the raw images to extract the mouth information. However, it was shown that optical ASR greatly increased recognition rates compared to acoustic recognition alone.

The next generation of optical ASR systems were more dynamic in that they evolved from using still images as templates to a system that tracked lip motion [2]. Windows were placed at strategic locations surrounding the mouth (one at center of top lip, one at center of bottom lip, and two at corners of mouth). Algorithms were then used to calculate the mouth motion within each window. The two results, mouth elongation and mouth opening, as functions of time were utilized as features. As before, good recognition results were obtained.

1.2 Objective Statement

The objective of this thesis is to determine if additional information obtained from IR or visible detectors can be used to increase the recognition rate of an audio ASR system.

1.3 Approach

The first step in this project will be to create an analog circuit that accurately records lip movement using IR or visible detectors. The circuit will prepare the signal for further digital processing. However, several designs may be evaluated before the analog signal can be digitized. This is because a reliable and useful signal must be found which meets the following specifications for photo ASR [3]:

- The signal must reliably track mouth movement as a function of time.
- The signal has to be repeatable so that similar words have identical signals.

Once an adequate photo speech signal is found, the microphone's analog circuit will then be designed and built. This circuit will built along with the analog photo circuit on the same board.

The second step will be to design and build a two-channel A/D board that digitizes both the audio and photo signals from the output of the analog board. This A/D board will reside inside a PC. The converted signals will then be stored as raw binary data on a hard drive inside the computer.

The last step will be to develop the software tools necessary to interface with the A/D board and process the data for performing certain optical and audio ASR experiments listed below. In each case the main objective will be to obtain as high a recognition rate as possible.

1. Isolated word, speaker-dependent recognition using no sensor fusion information.

2. Isolated word, speaker-independent recognition using no sensor fusion information.

3. Isolated word, speaker-dependent recognition using sensor fusion information.

4. Isolated word, speaker-independent recognition using sensor fusion information.

5. Continuous word, speaker-dependent recognition using no sensor fusion information.

6. Continuous word, speaker-independent recognition using no sensor fusion information.

7. Continuous word, speaker-dependent recognition using sensor fusion information.

8. Continuous word, speaker-independent recognition using sensor fusion information.

II. Analog System Designs

2.1 Passive IR System

For a purely passive system (i.e., no optical source), an IR detector should operate better than a visible detector for ASR. This is mainly because a visible detector is totally dependent on external lighting (e.g., the sun, office lights, etc.) whereas the IR detector only requires a source of radiation. Thus it was decided that a passive IR detector would be first tried for ASR.

The mouth acts as a heat source whose blackbody function peaks at about 9 um. Radiation from the mouth is contrasted by the areas surrounding the mouth which are at a cooler temperature. This can be seen in an IR video of a person talking. The ASR specifications for a single IR detector are listed below:

- Spectrum includes 9 um
- Field of view covers widest mouth opening (about 1 inch at 1/2 inch distance)
- Compact in size
- Have a response time faster than the mouth can move (about 25 Hz)

Several detectors were tested that met these requirements. One in particular had the best response with respect to mouth movement - a muRata ERIE IRA-F001 pyroelectric detector [4]. These detectors use crystal polarization to sense a change in temperature. An output voltage only occurs when there is a change in temperature. Consequently, most thermal measurement applications use a chopper to record the temperature of still objects [5:7]. However, a chopper should not be necessary in this effort since the object being measured (the mouth) will be in motion during an ASR recording.

Figure 1 is used to determine the optimum mouth-to-detector distance for the area covered by the detector's field-of-view.

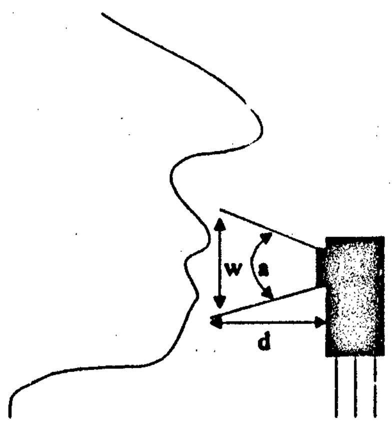


Figure 1. Geometry of thermal detector ASR test setup

The solution for "d" is shown below.

$$d = w/2/\tan(a/2) = 0.596 \text{ inches} \quad (1)$$

where

w = width of mouth at widest position (approximately 1 inch)

d = distance to mouth (inches)

a = detector field of view (80 degrees)

Since pyroelectric detectors are thermal devices, their wavelength response is theoretically unlimited. As a result, these are ideal for use in the far-IR (8-14 um) band.

It is interesting to note that these detectors are a close cousin of the piezoelectric microphone. Most manufactures try to eliminate one effect or the other in order to optimize a particular application (thermal or audio). Quite conceivably someone could design a single detector to simultaneously record both audio and IR information. Of course the data streams would be fused together and signal processing would have to be implemented to separate, if required, the two signals.

The electronic circuit used to interface with this detector is shown in Figure 2.

This is a simple bandpass amplifier circuit with f_l (low frequency cutoff) at about 1/2 Hz and f_h (cutoff frequency) at about 5200 Hz.

The circuit along with the detector was mounted on a PC breadboard. The speaker had to lean over and talk into the detector and maintain about 1/2 inch distance in order to generate a usable speech signal. The output of the amplifier was measured using a strip chart recorder which can be seen in Figure 3.

It is quite obvious from Figure 3 that this signal does not meet the photo ASR requirements - a repeatable signal that reliably tracked mouth motion. Due to the AC coupling in the circuit and the inherent nature of the pyroelectric detector, a signal occurred only while the mouth was moving. However, this test clearly demonstrates that it is possible to detect the boundary between words. This raises hope that some useful photo system may be found. Instead of trying to improve this circuit, it was decided that an active visible system would be tested next.

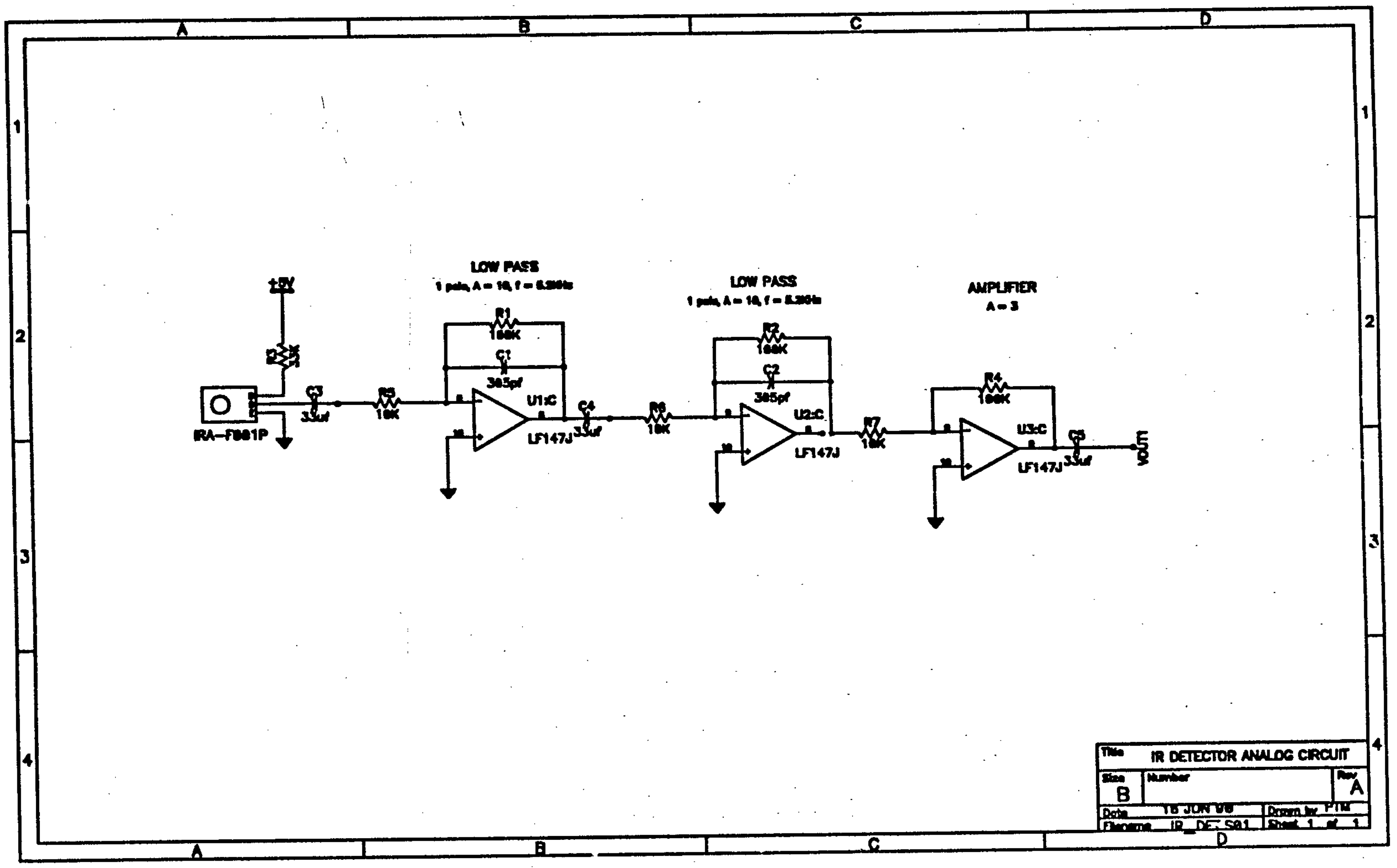


Figure 2. IR Detector Circuit

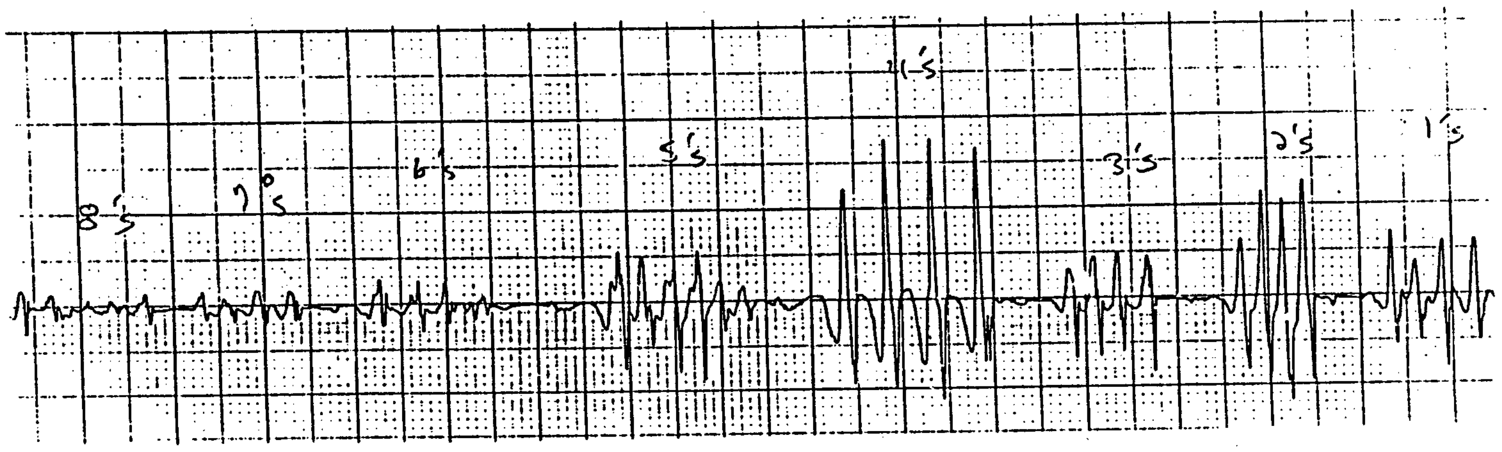


Figure 3. IR speech signal (multiples of words 1-8)

2.2 Active Visible Systems

An active visible ASR system is one that incorporates a LED as a source of radiation and a photo detector to receive the signal. The idea is to illuminate the mouth with the LED and have its reflected signal sensed by the detector. Unlike the thermal detector, a photo detector detects photons directly. This is accomplished through the interaction of photons and the P-N junction of the semiconductor. Arriving photons of sufficient energy will excite electrons across the energy gap. The resultant voltage potential will be a function of the amount of light that arrives at the surface of these semiconductors [6:117].

One of the lessens learned with the IR circuit was that head movement caused the signal to not be repeatable. To compensate for head movement, the visible detector was mounted, along with the LED and microphone, on a head set.

Another problem with the previous IR circuit was that without a chopper, no constant voltage output would occur for a constant thermal input. Eventually the voltage signal would peak and go back to its baseline. Using AC-coupling would have had the same effect. This configuration is not optimal because it is important that the detector's output be a function of mouth position. A system had to be designed that would output a small voltage (around zero) when the mouth was closed and a large, "constant" voltage when the mouth was opened and remained open (or vice versa).

There are at least two ways to meet these requirements while using an active LED as the source and a photo detector as the receiver: (1) use DC coupling for the entire circuit or (2) use AC coupling with an AM (Amplitude Modulation) signal whose modulation is a function of mouth movement. The advantage of DC-coupling is "simplicity". All that is required is filtering, amplification, and offset adjustment. Unfortunately, DC offset adjustments can be cumbersome if the signal is not well-behaved. For instance, if there are any sizable offset noise shifts, such as headset movement, the photo signal could saturate the electronic amplifier circuits and never be seen at the output.

The most attractive aspect about AM is that the signal requires no offset adjustment. However, either the hardware or the software has to perform additional signal processing to extract the carrier's envelope. Usually an envelope detector is implemented to perform this operation, in addition to the filtering and amplification that the circuit must also accomplish.

The AM photo system was chosen, over the DC system, to be evaluated next. It was feared that there would be excessive noise caused by headset movement. This was due to the fact that the headset was not lightweight and was somewhat uncomfortable.

2.3 Active Photo AM System: A diagram of the proposed AM system is shown in Figure 4. The theory of operation is to generate a LED CW (continuous wave) signal that is amplitude modulated by the mouth. The circuit used an oscillator to generate a constant frequency square wave which was sent to a LED to generate the CW photo signal. Since the LED rectified the signal, there was very little difference between using a square wave instead of a sine wave. The signal was then modulated by the mouth and detected by a visible-to-near-IR photo detector whose output was subsequently filtered and amplified. Signal processing was accomplished with software to detect the envelope.

The detector and LED was located fairly close to the mouth (about 1/4 to 3/4 inches) to obtain as high a S/N as possible. Some noise arrived from surrounding artificial light sources. To counter this, the CW frequency was adjusted away from the frequencies of some noise sources such as 60 and 120 Hz.

Advanced Optoelectronic's 44PH05M near IR photodiode was chosen as the detector to receive the AM signal [7]. Its large active detection area (17 mm^2) helped increase the S/N. Likewise, the 44PH05M's peak wavelength occurred at about 950 nm which also helped increase S/N since this specification was located outside the fluorescent light spectral range and at a dip in that sun's spectral absorption band [6:116].

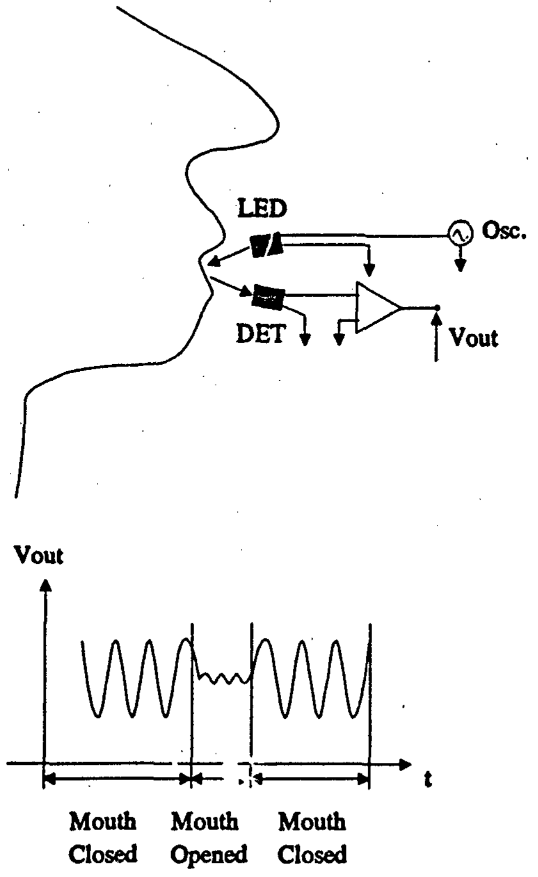


Figure 4. Diagram of proposed AM photo system

To transmit the light, a Radio Shack Super-Bright LED was used. This was one of the brightest yet smallest LEDs sold by Radio Shack.

There are basically two modes of operation for photodiodes - photovoltaic and photoconductive. The main difference between the two modes is biasing - the photoconductive mode uses biasing and the photovoltaic mode does not. Another difference between the two is that the photoconductive mode has a faster response time. However, it is inherently noisier due to an increase in shot noise [6:119].

A 450 nsec response-time specification for the photodiode was quoted for the photovoltaic mode. This is very good since the CW frequency was adjusted to around 279 Hz which is well below both this specification and the LED's rise and fall times of about 1.0 usec. As a result, a fast response time for the detector was not critical and therefore the photovoltaic mode was selected for this effort.

A schematic of the circuit is shown in Figure 5. A photodiode is basically a current source that when illuminated, generates an output that is proportional to the light intensity. To convert the current to a voltage, a transimpedance amplifier was used in the first stage. The voltage signal was then filtered and amplified which was previously checked by sweeping a sinusoidal waveform through its bandwidth and observing the results on an oscilloscope.

Since the A/D board sampled the photo data at about 2.5 KHz (f_S), the signal's highest frequency cannot exceed $f_S/2$ or about 1.25 KHz. Taking into account system noise, a good figure-of-merit is $f_S/5$ or 500 Hz. To be safe the CW frequency was adjusted, as mentioned earlier, to around 279 Hz.

As can be seen in the schematic, the bandpass filter frequencies are 200 and 400 Hz. The output of the bandpass was then amplified twice before being sent to the A/D board. Since the CW frequency was less than $f_S/5$, aliasing was not a problem and, as a consequence, a software algorithm was used to extract the signal's envelope.

The headset and analog circuit were adjusted for as large an amplitude signal as possible that was within the A/D's +/- 2.5 volt range. Also they were adjusted to achieve the largest mouth open to mouth closed signal as possible. A photo of the head set after adjustments, is shown in Figure 6. As can be seen, this is a crude version of commercially available microphone headsets. An additional benefit to using the Radio Shack LED was that it emitted a visible red spot on the mouth that was used during alignment of the headset.

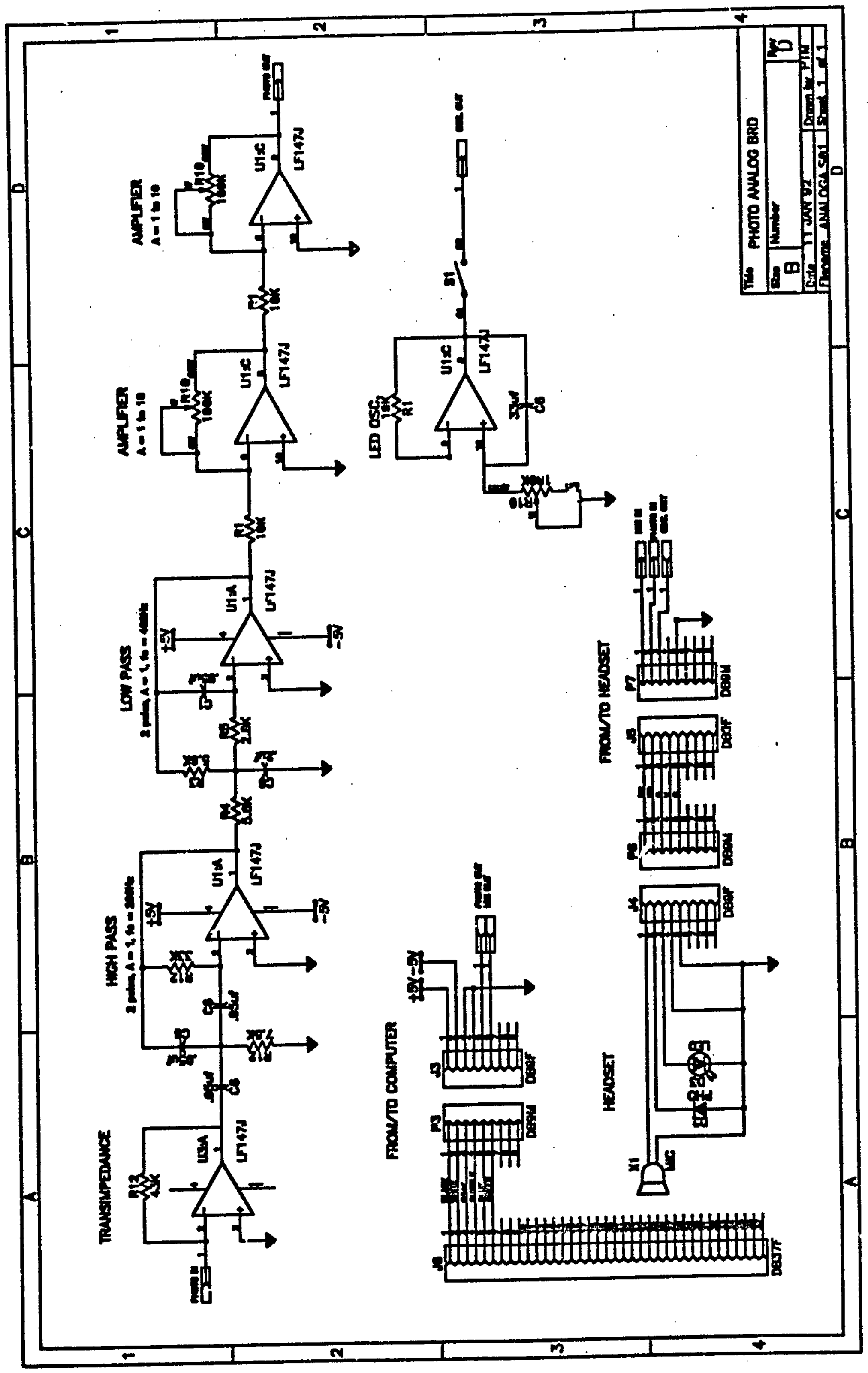


Figure 5. AM photo circuit

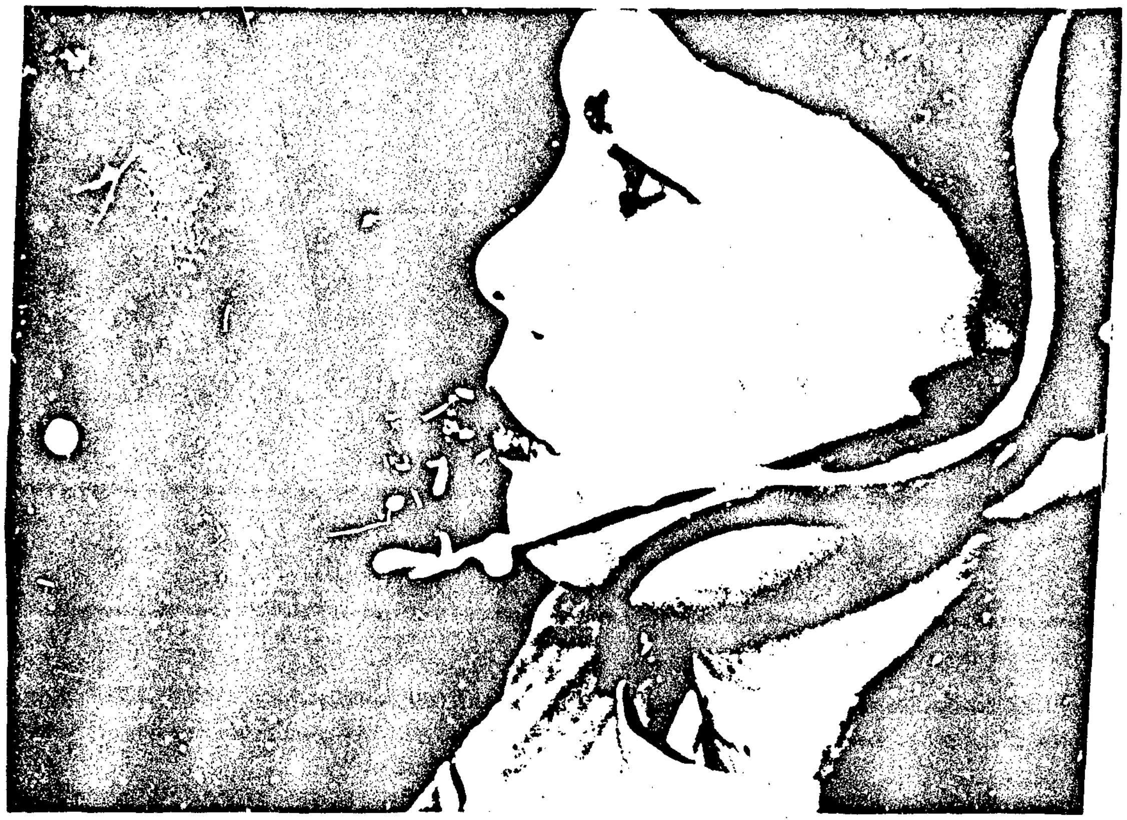


Figure 6. Photograph of audio/photo ASR headset

An example of the word "one" for the received AM photo signal is shown in Figure 7. Both the time and frequency domains are shown. The small peak at 120 Hz was due to ambient light noise. As can be seen, the S/N was very good. From this figure, simple mathematical equations describing the AM signal can now be derived.

For this discussion we will assume that only a single sinusoid signal is emitted from the output of the LED. This should be valid since the harmonics of the actual square-wave were filtered-out in the analog circuit.

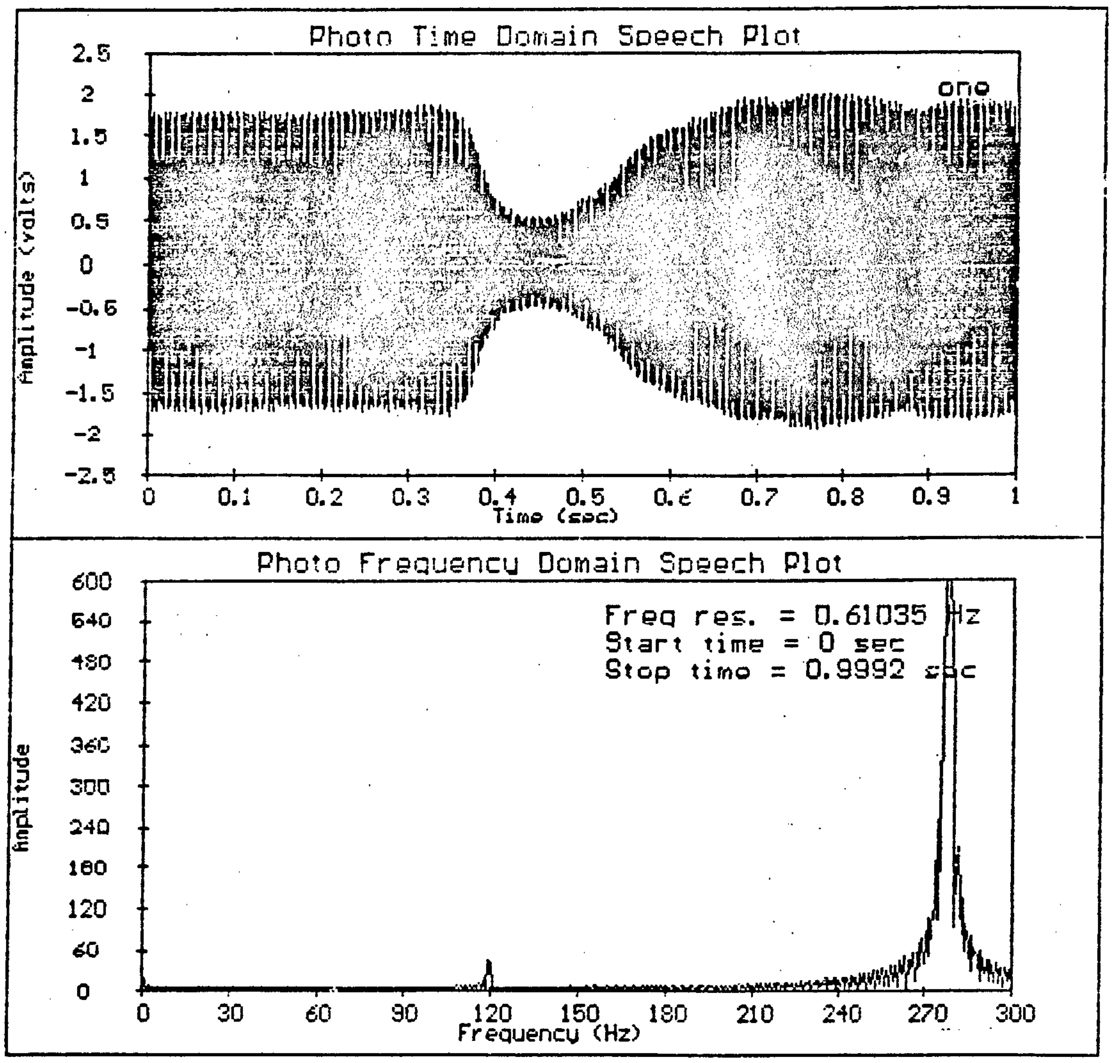


Figure 7. Example of the word "one" for AM photo signal

A pictorial description of the components of the active photo setup can be seen in Figure 8.

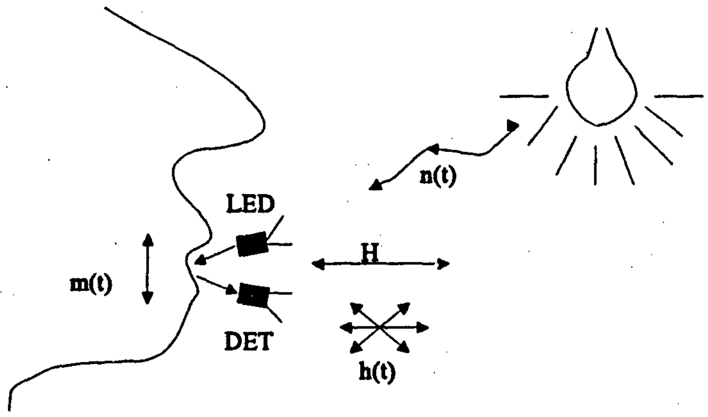


Figure 8. Parameters used in AM photo equation

The basic AM equation for the above figure is shown below

$$f(t) = K*[m(t) + h(t) + H]*\cos(w_C*t) + n(t) \tag{2}$$

where

$f(t)$ = resultant photo signal received at detector

$m(t)$ = mouth movement signal

$h(t)$ = headset noise signal

H = constant DC offset that is a function of the emitter/receiver distance to mouth.

w_C = emitter (carrier) radian frequency

$n(t)$ = ambient light noise from external sources

K = LED oscillator gain

Photo ASR information is contained in "m(t)" - the mouth modulation signal. It is the main signal of concern for which all signal processing (both hardware and software) was designed around. The majority of the amplitude for "f(t)" came from "H" which was, up to a certain distance, inversely-proportional to the distance from the headset to the mouth. When a certain minimum distance was reached, most of the signal did not reflect back into the detector and the amplitude of "f(t)" started to decrease. Increasing the headset-to-mouth distance had the same affect - after some optimum distance, "H" started to decrease. This relationship defines "H" as a non-linear geometric function of the mouth-to-headset distance.

The ambient light noise term is "n(t)" which could be light received from the sun and/or, as we saw in an earlier figure (see Figure 7), artificial lighting. The affect of this noise term was minimal since the S/N was high when "H" is adjusted for as large amplitude as possible. The most significant noise term was "h(t)" which was generated when there was head movement or, in other words, a change in the headset-to-mouth distance. The affect on the resultant AM signal "f(t)" was an amplitude change. Unfortunately "h(t)" was the hardest signal to filter-out due to its signal content (frequency and amplitude information) being approximately that of the "m(t)" signal. This was determined after some experimentation with the headset and analog circuit whose output went to a spectrum analyzer.

The resultant signal, after hardware and software processing, contained both "m(t)" and "h(t)" terms. Most of the "n(t)" noise term was filtered out by the analog circuit of Figure 5. But the "h(t)" noise term still existed even after filtering. However, it was determined after some trial runs that under static conditions (i.e., the subject was sitting down and not moving around), the "h(t)" term goes to a very small value and was not significant.

2.4 Microphone system: A schematic of the microphone circuit is shown in Figure 9. A Radio Shack PC-Mount Condenser Microphone Element was used as the audio transducer. It was mounted to the side of the photo detector as shown in the photograph of Figure 6. The microphone's output was amplified between 20 to 420 times and then bandpass filtered. Since most of the frequency spectrum for audio lies below about 5 KHz [8:33], the highpass cutoff was chosen to be 5 KHz.

Since the microphone's output was also its power supply input, power supply noise can easily be coupled into the signal. Unfortunately a PC's power supply is very noisy. As a result, a 9V chemical battery was used to power the microphone instead of the PC's internal +/- 5V electrical power supply.

An example of the audio word "zero" is shown in Figure 10. The periodic noise that is seen before the actual word was due to cross-coupling of the photo CW signal into the audio signal. The frequency of the photo signal was purposely decreased so that its coupled period could be observed. The source of the noise was traced to the flat-ribbon cable whose lines carried both the photo and audio signals, in parallel, to and from the headset. This was a serious problem because it required replacing the current light-weight ribbon cable with shielded lines such as RG-188's. The end result would have been a bulkier, heavier setup that would probably be more uncomfortable thus increasing headset movement noise "h(t)" for the photo signal. Also there was no guarantee that the coupling noise would have been completely eliminated. Since there are no clock signals used in the DC-coupled version, it was decided that this system would be evaluated next before a major overhaul of the headset was accomplished.

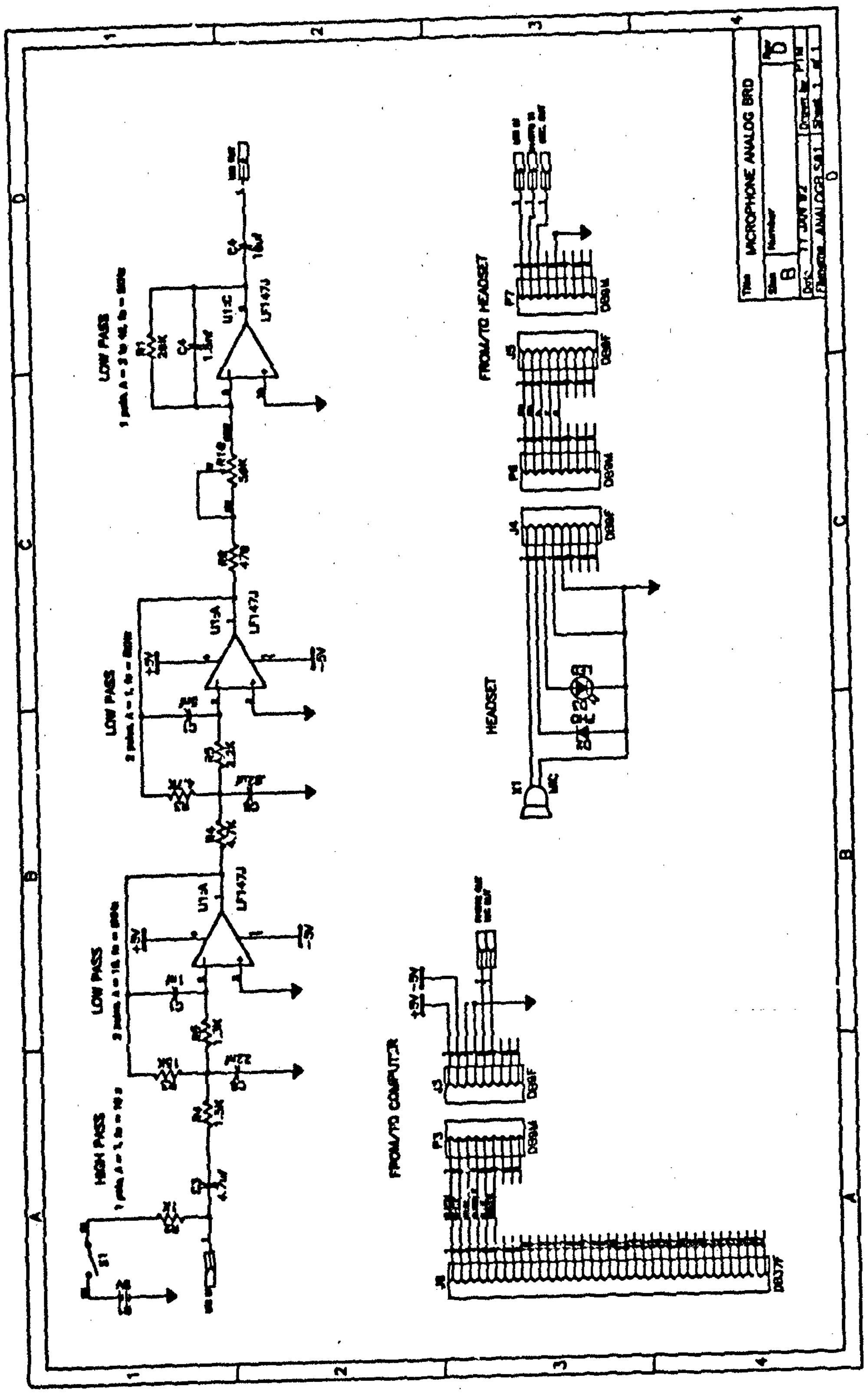


Figure 9. First version microphone circuit

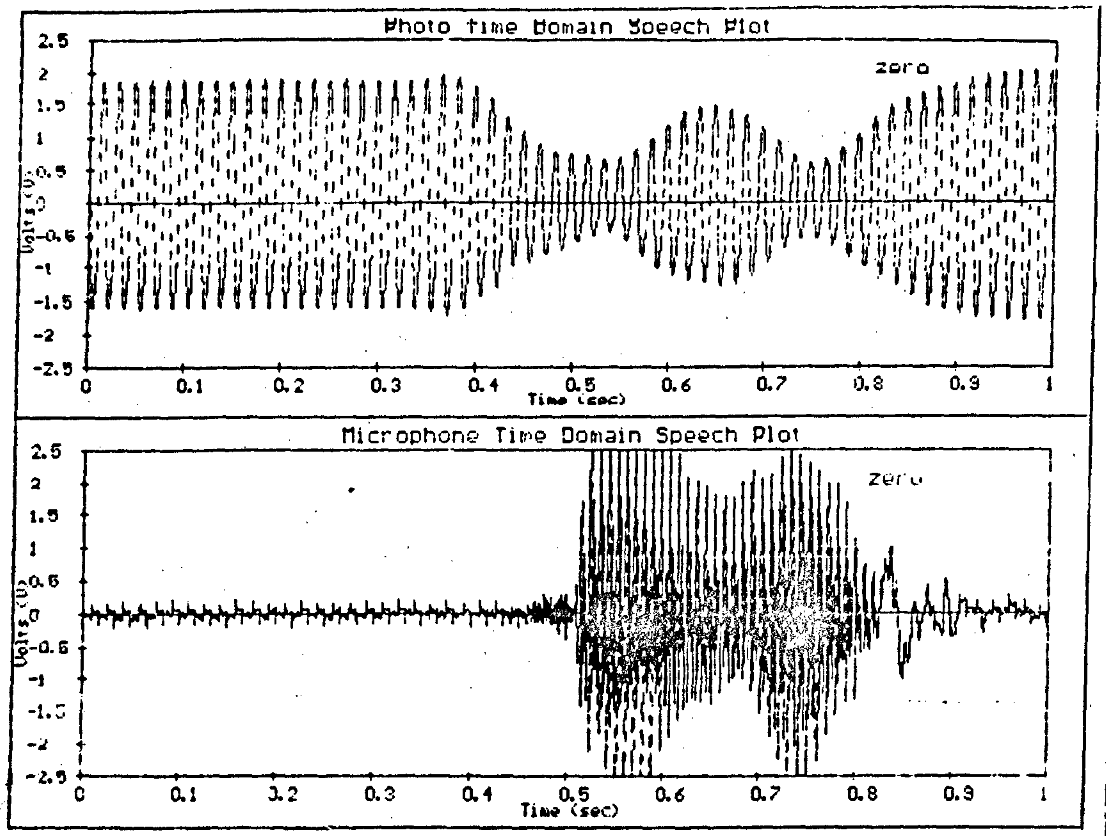


Figure 10. Audio signal for the word "zero"

Before going on to the next section, note the signal noise between about 0.8 and 1.0 seconds in Figure 10. This was caused by reflections in the transmission line due to impedance mismatch between the analog circuit and the data acquisition board. The solution was to put line drivers and receivers at each end of the transmission line. The new microphone interface circuit shown in Figure 11 contains a line driver. With a few exceptions, this circuit was basically the same as the previous microphone circuit.

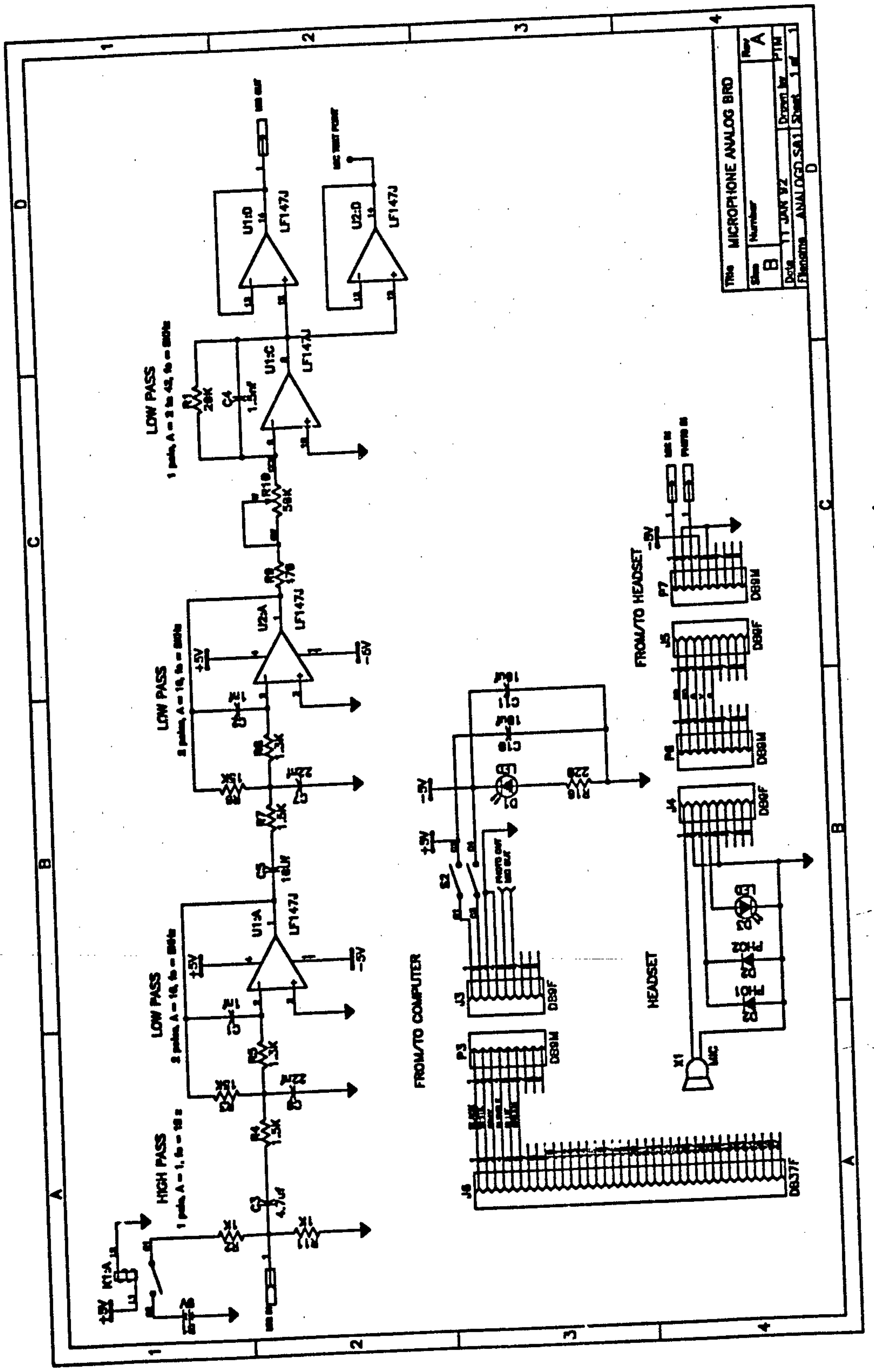


Figure 11. New microphone circuit

2.5 Active Photo DC-Coupled System: Silicon Detector Corporation's (SDC) SD-020-11-11-011 red/IR photodiode was chosen as the detector for the DC-coupled photo system [9]. In addition, their super high-output GaAlAs IR emitting LED was chosen to be the emitter. Both devices' have their peak wavelengths matched at about 880 nm. A diagram of the new setup can be seen in Figure 12.

The new "f(t)" signal, which is shown below, was similar to the AM signal but without a trigonometric carrier term.

$$f(t) = K*[m(t) + h(t) + H] + n(t) \qquad (3)$$

where

$f(t)$ = resultant photo signal received at detector

$m(t)$ = mouth movement signal

$h(t)$ = headset noise signal

H = constant DC offset that is a function of the emitter/receiver distance to mouth.

$n(t)$ = ambient light noise from external sources

K = LED gain

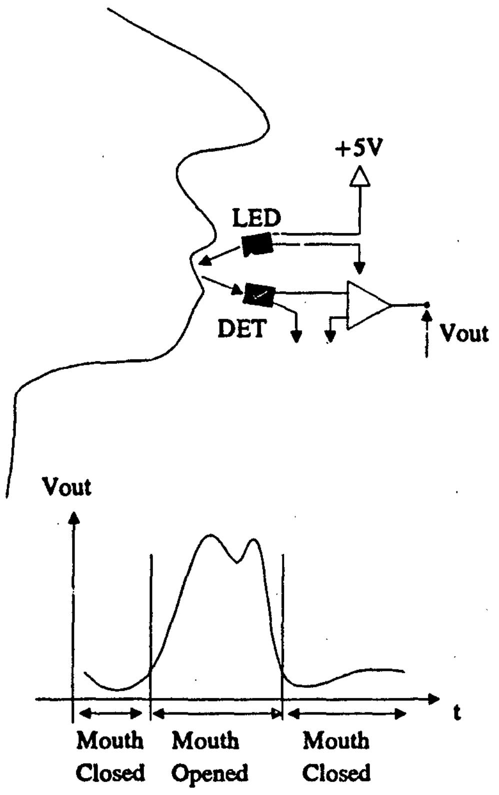


Figure 12. Active visible ASR setup

The fastest "m(t)" signal that the photodiode detected should be around 25 Hz [3]. Consequently the photodiode's response time of 12 ns and the LED's rise and fall times of 0.5 usec were overly sufficient. Also, the lowpass filter cutoff for the analog circuit will be set at about 25 Hz.

The LED's peak spectral wavelength was near-IR which is located outside the fluorescent light spectral range and between the sun's and tungsten peak wavelength ranges [6:116]. This helped increase the S/N by moving the modulation signal's wavelength further away from the ambient light noise's "n(t)" wavelength bandwidth. Black hoods made out of heat shrink were used to help eliminate ambient light noise. Unfortunately these hoods also narrowed the field of view. This was in addition to the photodiode's already small detection surface area (0.2 mm^2 compared to 17 mm^2 for the previously used Advanced Optoelectronics 44PH05M). To increase the collecting area, two SD-020-11-11-011 matched detectors were used. One was mounted on top of the LED and the other was mounted on the bottom of the LED. A photograph of this new setup is shown in Figure 13 and its geometry will now be discussed. The next figure,

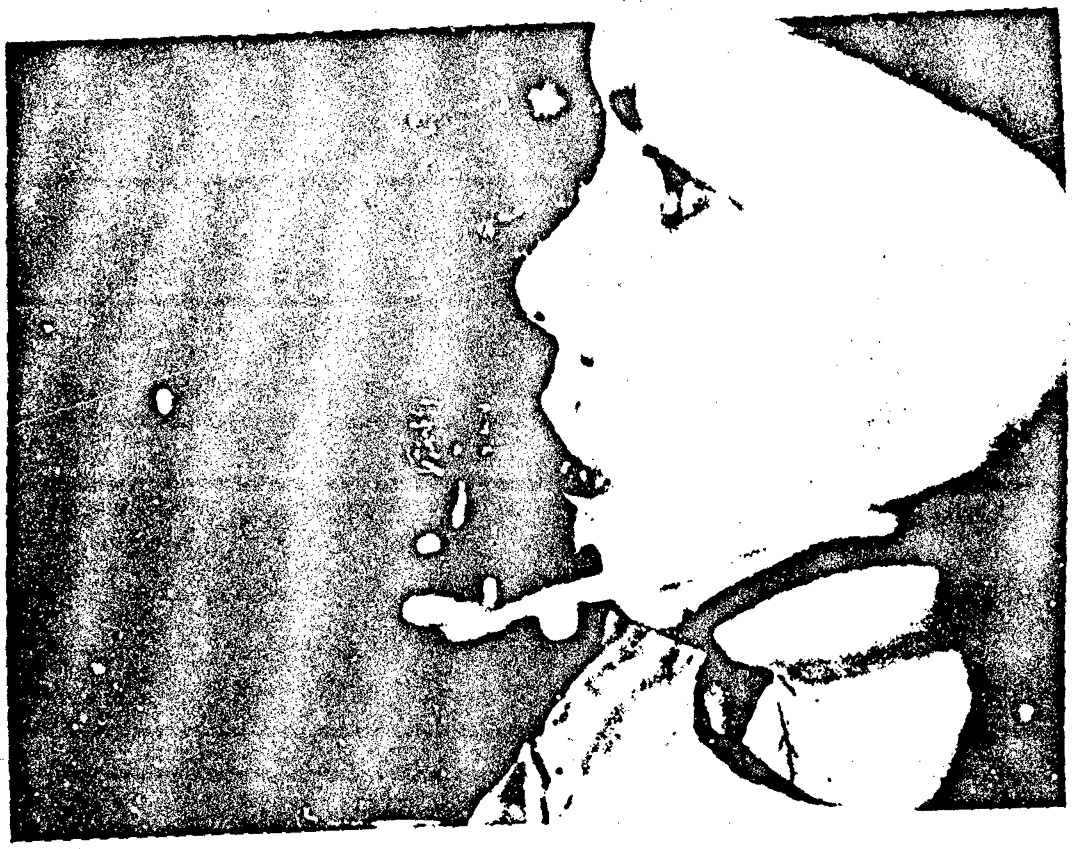


Figure 13. Photo of new headset setup being worn

Figure 14, is a graphical description of the geometrical parameters involved with the new photo ASR system headset.

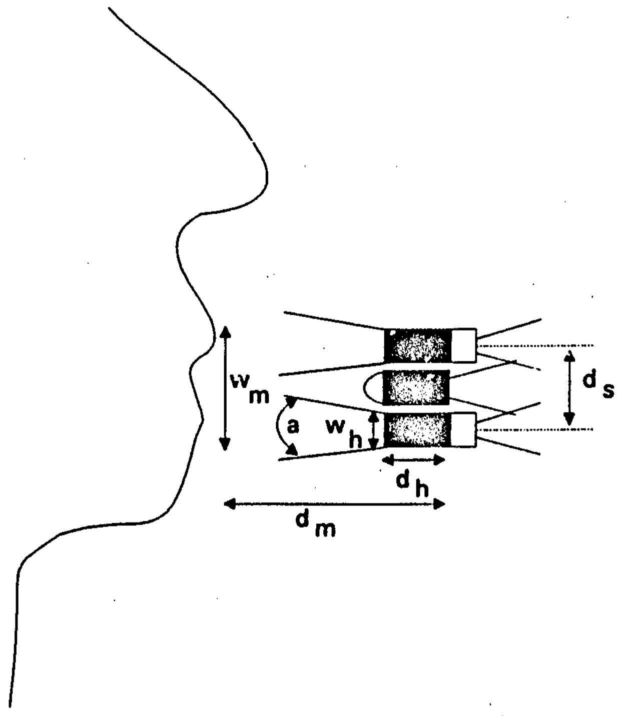


Figure 14. Geometry used for DC-coupled active photo ASR system

Where the parameters in the figure are defined below as:

w_h = width of hood opening (about 3/16 inches)

a = field of view

w_m = distance at mouth covered by field of view

d_h = length of hood (about 6/16 inches)

d_m = distance from mouth to end of hood (about 3/4 inches)

d_s = separation distance of detectors (about 6/16 inches)

The field of view for Figure 14 is solved below.

$$a = 2*\tan^{-1}(dh/2/dm) = 28^{0} \qquad (4)$$

The distance at the mouth covered by a single detector's field of view was:

$$w_m = 2*d_m*\tan(a/2) = 0.4375 \text{ inches} \qquad (5)$$

Taking into account "d_S", the approximate total distance covered at the mouth for both detectors was about 0.8 inches. There was some overlap depending on the distance. In general, a distance between 1 and 1/2 inches resulted in an optimum response with respect to lip movement.

An additional benefit of doubling the active receiving area was an increased S/N. Furthermore, most of the signal received came from the LED source since a majority of the "n(t)" term was blocked-out by the black hoods.

The circuit used for the DC-coupled photodiodes is shown in Figure 15. Like the previous photo circuit, the first op-amp was used as a transimpedance-amplifier. However, unlike the previous circuit, it is also a low-pass filter at about 5 Hz. A single pole at this frequency did dampen the "m(t)" signal but, more importantly, it helped eliminate noise terms. The second amplifier is a low-pass filter at about 25 Hz. In addition, it was used to adjust the DC offset and amplify the resultant signal.

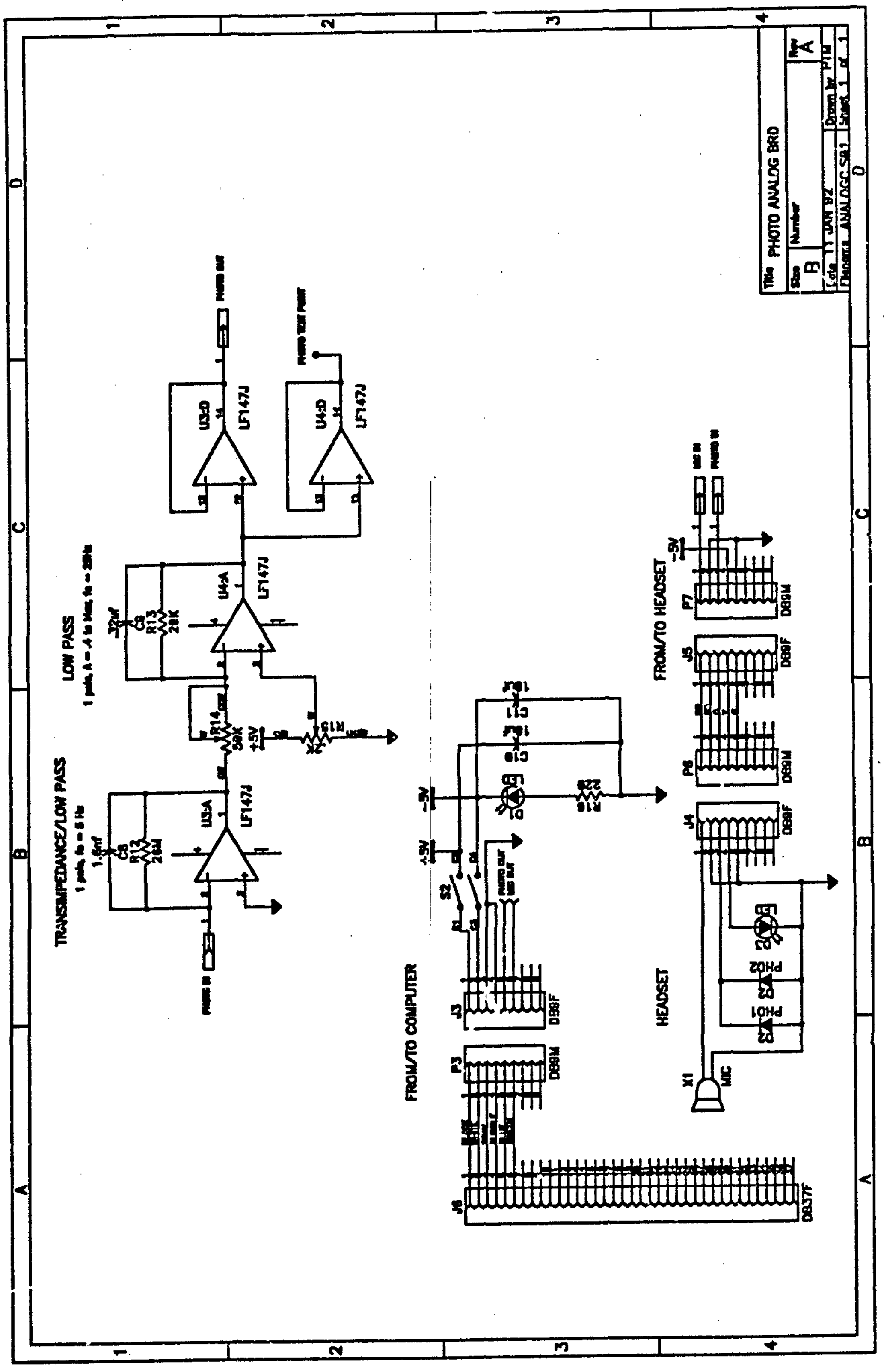


Figure 15. DC-coupled photo circuit

An example of a DC-coupled photo waveform is shown in Figure 16 for the word "eight". This was obtained by adjusting the offset of the signal to about -2.0 volts. The amplitude was then adjusted for a maximum of about +2.0 volts using the word "five" due to its wide response. These alignments put the photo signal within a fairly good portion of the A/D's dynamic range (+/- 2.5 volts). The resultant signals met the photo ASR requirements and did not affect the audio signal. Consequently, the DC-coupled, active-photo system was chosen to conduct photo ASR experiments for the rest of this effort. The audio and photo signals were then digitized by a data acquisition board which was designed and built during this project. Refer to Appendix A for the details of this board.

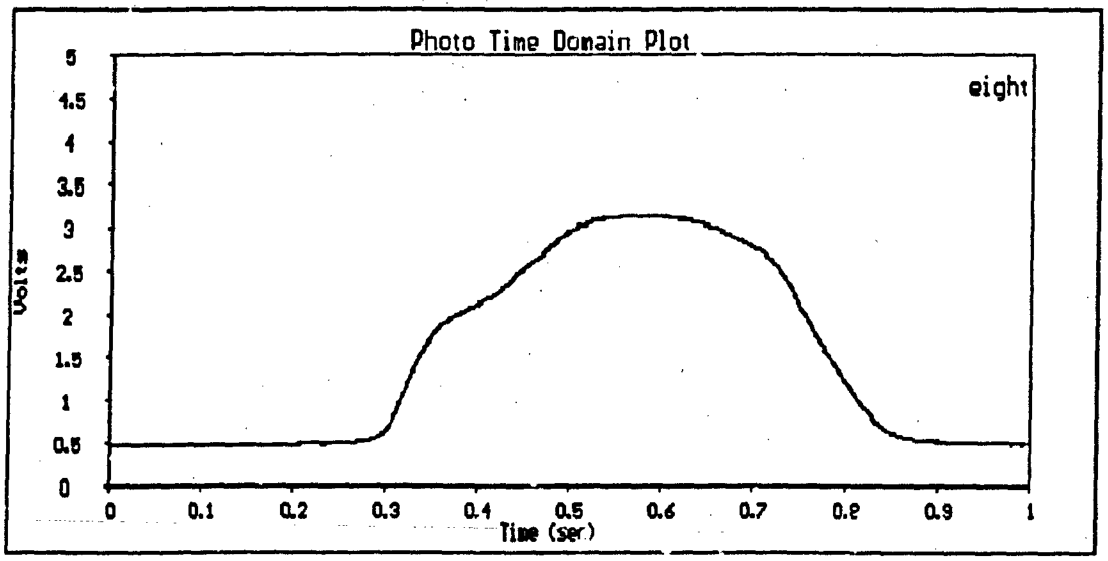


Figure 16. DC-coupled photo signal for the word "eight"

III. Data Collection

Five people were tested: two males over 30 years old, 2 females over 30 years old, and one 7-year old child. Each subject spoke 12 isolated words and 5 continuous words made up of the isolated words. A list of these are shown in Table 1.

Table 1. List of isolated and continuous words used in experiments

Isolated Words	Continuous Words
zero	seven-one-one-one
one	one-nine-one
two	yes-eight-nine-no
three	no-four-seven-yes
four	two-eight-two-eight-two-eight
five	
six	
seven	
eight	
nine	
yes	
no	

A word scrambling algorithm was used for the isolated words. This was necessary to keep the test subject from getting comfortable with saying the same words in the same sequence each time.

The continuous words were made up of isolated words. Several continuous word sequences were tried with the objective of decreasing the audio recognition rate using no sensor fusion information. The ones listed in column two seemed to have the lowest audio recognition rates.

A total of 5 multiples was made for each isolated word and continuous words. This gave a sum total of 60 isolated (12 * 5) runs and 25 continuous (5 * 5) runs for each person.

After some alignment of the headset and adjustment of the electronics, each person sat in front of the computer screen where, after pressing a key, either an isolated word or continuous words would appear. The adult persons then had 1.0 second to say the isolated word and 2.5 seconds to say the continuous words. The one child (Jill) had 1.5 seconds to say the isolated word and 3.5 seconds to say the continuous words. After the allotted time was up, the algorithm then down-loaded the data to hard disk and prompted the speaker to hit any key to continue. The next word would then pop-up at the screen and the process repeated itself. The entire data-collection run took about 15 minutes for each person.

IV. Speech Processing Algorithms

This section describes certain speech processing algorithms used in this thesis. An overview of the steps involved with processing the templates and conducting isolated and continuous recognition are list below.

• Both types of words (i.e., photo and audio) were located.

• Both types of word boundaries were verified by comparing their endpoint locations (template and isolated sensor fusion only).

• Both types of words were then segmented and data was reduced.

• The audio words were then transformed into the frequency domain and filter-bank processing was applied to the results.

• Both types of words were then normalized.

• Both types of words were then compared to all templates to obtain a match.

• Finally, both types of words had their recognition results compared to decide which was correct (isolated and continuous sensor fusion only).

The algorithms in this section are also described in Appendix B with respect to actual software programs and functions included in Appendix C. Refer to Appendixes B and C for more information related to the software details of this project.

4.1 Audio Energy Distribution: To locate audio word boundaries, the microphone's energy distribution in the time domain was used. However, unlike the photo signal, the audio's average magnitude had to first be calculated. This was accomplished through the use of a rectangular window that summed the absolute values within a given bin size and computed the average value. The bin size determined how many data points were averaged from the audio's original voltage signal for a given window of time. The total number of bins was dependent on the sample's time length, sampling rate, and bin size. For instance, if a sample's length was chosen to be 1 second, and the bin size was chosen to be 500 samples, then the resultant energy distribution would be 50 bins in length. This is a function of the audio's sampling rate which was 25 Ksamples/sec (25000/500 = 50). Each bin result would then be placed in time half way between the beginning and ending window time used to calculate that particular bin's average value. Using the same example, the result of the first bin would be placed at 1/100 seconds assuming the signal started at zero seconds. It was very important to keep the time information correct since these results were used to locate audio words and eventually photo words.

Unfortunately pitch variation is so significant (between 80 and 160 Hz for males and between 160 and 400 Hz for females [10:98]) that there can never be an optimum bin size for everyone's audio energy distribution. Consequently some experimentation with the bin size had to be accomplished to obtain a single bin size for all subjects tested. Figures 17 and 18 are audio energy plots with the bin size, "N", equal to 250. However

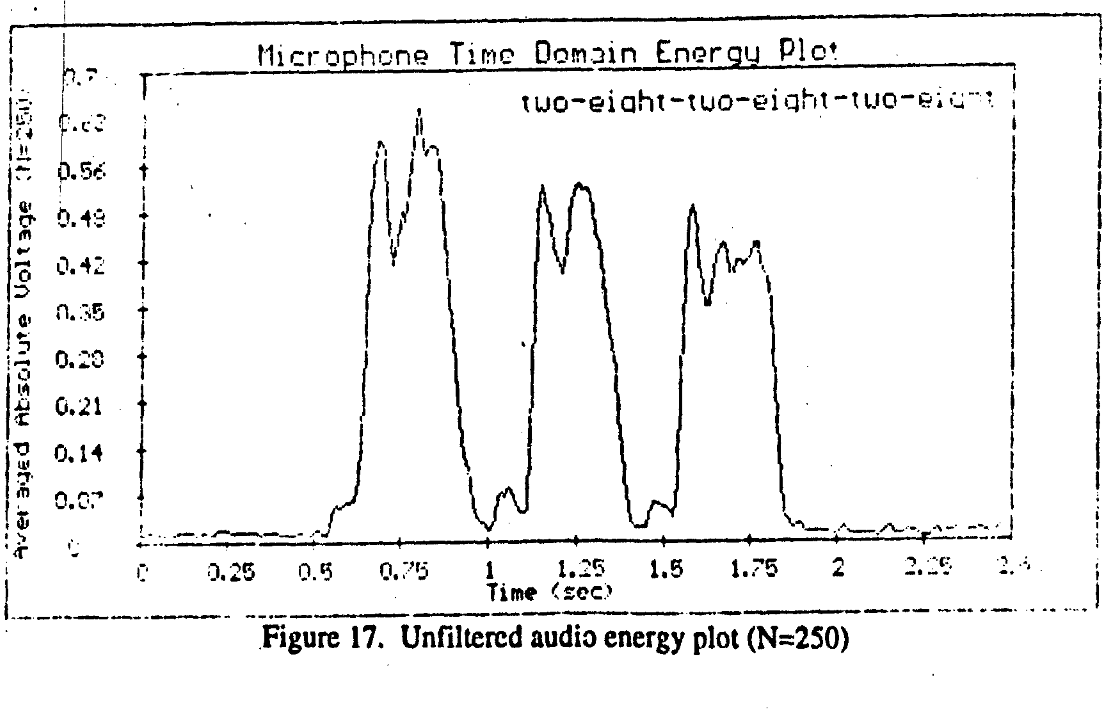


Figure 17. Unfiltered audio energy plot (N=250)

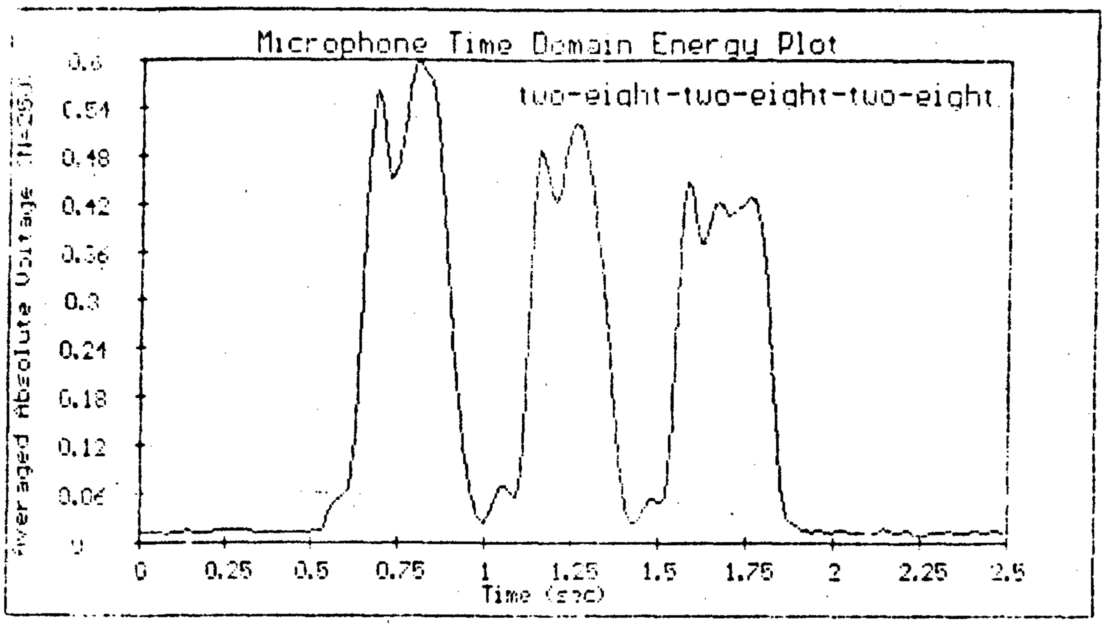


Figure 18. Filtered audio energy plot (N=250)

Figure 18 was filtered using a five-channel smoothing algorithm which is shown below [11:257].

$$y''(t) = 1/16y(t-2) + 1/4y(t-1) + 3/8y(t) + 1/4y(t+1) + 1/16y(t+2) \qquad (6)$$

It is quite apparent from the comparison of these two plots that the smoothing algorithm helped eliminate false word boundaries while maintaining real word boundaries. However, more experimentation with the bin size "N" was necessary to determine if these results could be improved upon.

Figure 19 is the audio's energy distribution for N = 300 and Figure 20 is with N = 400. Both functions were filtered by Equation 6. From these plots, and Figure 18, it was decided that N = 300 had the best results since the word boundaries for "two and "eight" could easily be distinguished while simultaneously eliminating most of the false word

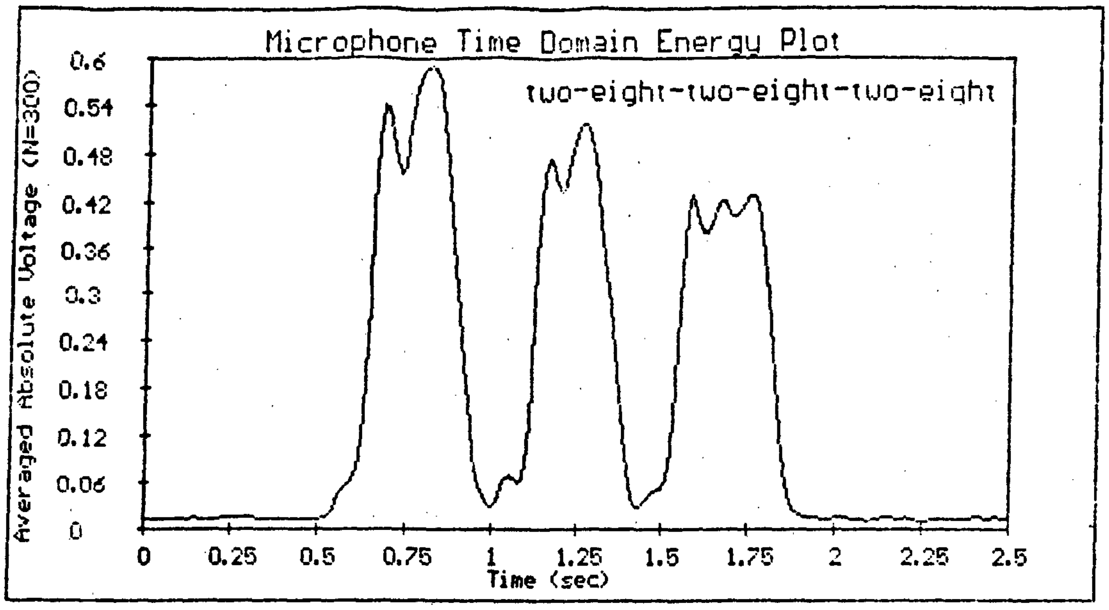


Figure 19. Filtered audio energy plot (N=300)

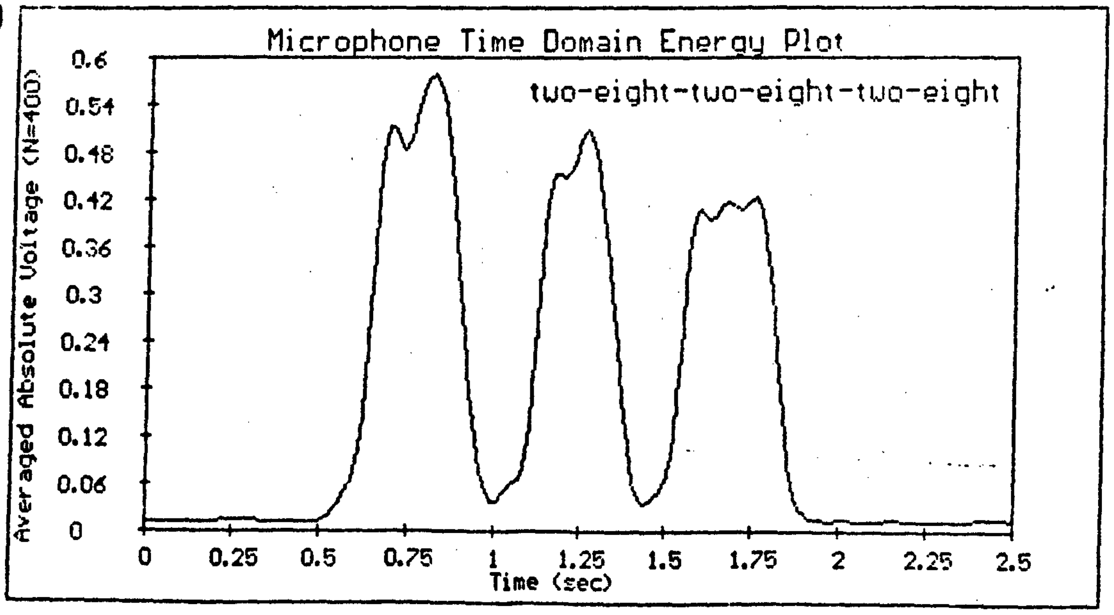


Figure 20. Filtered audio energy plot (N=400)

boundaries. This result was complemented by one author's recommendation that a bin size of 100-200 for a 10 KHz sampling rate was a suitable choice for distinguishing between voiced and unvoiced regions [12:122]. This was comforting since the audio was, for this project, sampled at 25 KHz.

Finally, this bin size was verified with the other four subjects and other continuous words and was found to be satisfactorily with respect to highlighting word boundaries.

4.2 Locating Words: As previously mentioned, the audio's energy distribution was used in this effort to locate audio words. Likewise, since previous experiments using camera recognition found that word boundaries occurred when the mouth was opening, reversed direction, or simply stopped, the photo's raw voltage signal as a function of time was used to locate photo words [2:4].

An algorithm was developed that sensed huge changes in amplitude for the purpose of locating both photo and audio words. This was chosen since word boundaries are somewhat a function of changes in signal levels. The algorithm incorporated a gradient search method to locate slopes of sufficient magnitude which were assumed to be word boundaries. When completed, this algorithm would return the beginning, ending, and peak times along with the peak value of every word found.

Sensor fusion was incorporated to help find both types of words. Usually sensor fusion is a multiple device system, each of which produces an output independent of the other. The information from the different devices is then analyzed (i.e., "fused together") to obtain a best guess [3]. In other words, sensor fusion usually occurs at the end of the recognition process. This traditional view of sensor fusion was modified for this effort since there was additional useful information gained from the comparison of word boundaries. As a result, sensor fusion was used prior to the recognition process to verify the location of photo words for both template processing and isolated word recognition. In addition, the photo's word boundary information was the only boundary information used for both signals during the continuous word recognition experiments using sensor fusion information.

4.2.1 Post processing word boundary verification algorithm used for isolated and continuous word recognition using no sensor fusion information: The results returned by the gradient search algorithm for both the isolated and continuous cases was not processed any further and was the only segmentation data used to conduct word recognition runs. No sensor fusion processing of any kind was programmed to enhance the results.

4.2.2 Post processing word boundary verification algorithm used for template processing and isolated word recognition using sensor fusion information: Audio information was not only used to locate audio words, it was also exploited to help verify isolated photo word boundaries for both template and isolated word processing. This form of sensor fusion was necessary since the human mouth may not be in the fully closed position prior to speech. Incorporating sensor fusion techniques during template processing was justifiable since it is very important to obtain the best word template possible. The microphone's signal was ideal to use for this purpose since the local environment was fairly quiet. As a consequence, most isolated audio words were not hard to locate. However, in harsher environments (e.g., airplane cockpits), this methodology may not be as successful.

After the gradient search algorithm returned its results, another algorithm would then process the word(s) boundary information to determine which word(s) was the correct word. This algorithm, whose steps are outlined below, used information from both signals (i.e., sensor fusion).

1. All audio and photo peaks were compared and matched. A match occurred when the distance between two peaks was less than 0.2 seconds. If more than one match occurred, the match with the minimum distance was chosen.

2. The audio word with the largest peak value was located and its corresponding word boundaries were initially selected as the correct audio word.

3. The corresponding photo word matched in step 1 to the audio word found in step 2 was then selected as the correct photo word.

4. If other audio peaks were within +/- 0.25 seconds of the audio word selected in step 2 then their endpoints were used as the new endpoints of the audio word selected in step 2.

5. The photo word's endpoints selected in step 3 was correspondingly adjusted to any new words found in step 4 that match up in step 1.

6. The photo boundaries are verified by comparing them to the boundaries of the audio word. If the photo's beginning endpoint was less than 0.1 seconds from the corresponding matched audio's beginning endpoint, then the photo's beginning endpoint was made equal to the audio beginning endpoint boundary minus 0.05 seconds. Likewise, if the photo's ending endpoint was greater than 0.1 seconds from the corresponding matched audio's ending endpoint, then the photo's ending endpoint was made equal to the audio ending endpoint plus 0.05 seconds. Figure 21 will now be used as an example of this entire process.

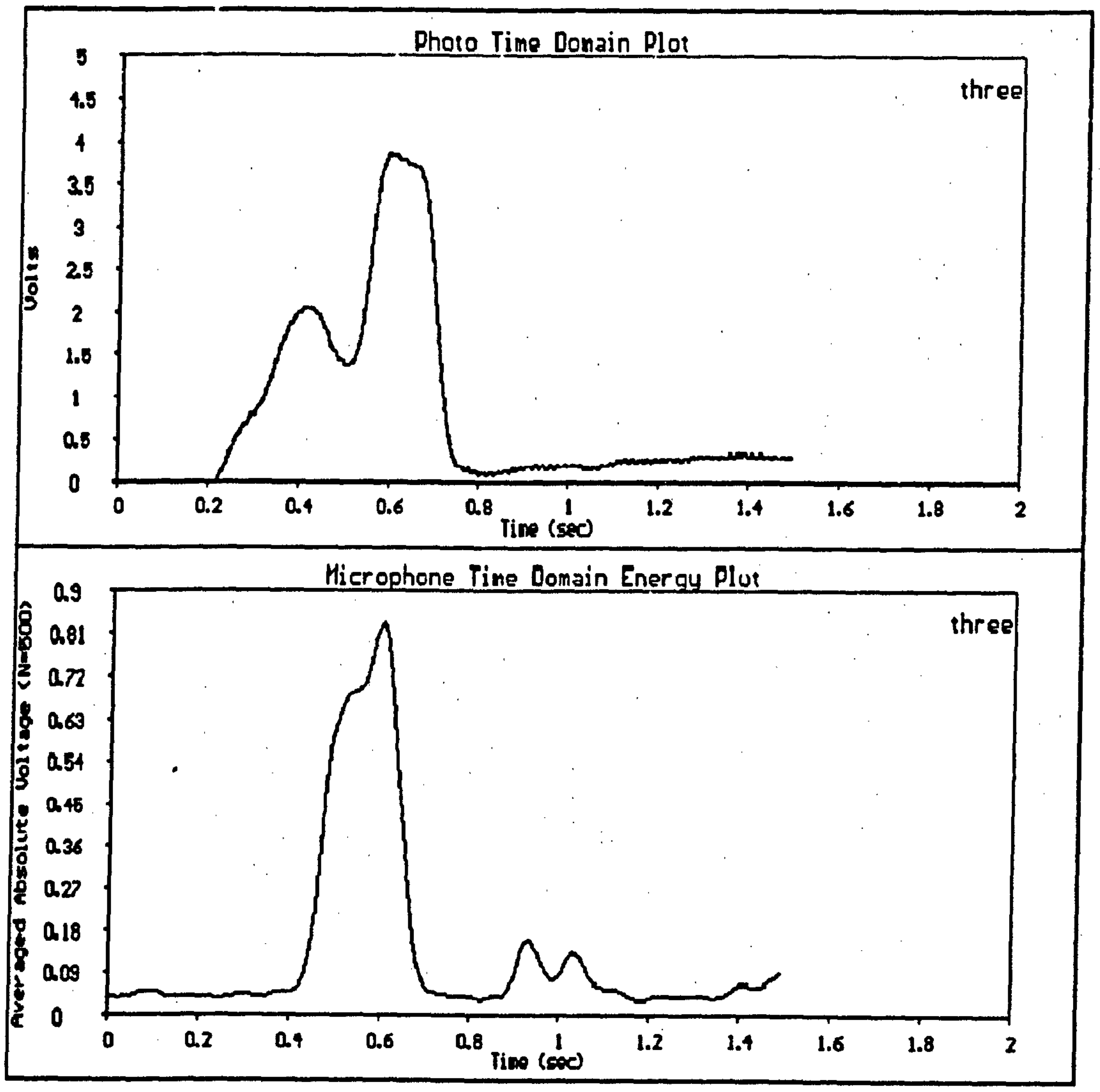


Figure 21. Example audio energy distribution being used to verify photo word

The gradient-search algorithms would have returned the following approximate information in Table 1 based on information given in Figure 21.

Table 2. Example of initial word segmentation results

Word number	Word Beginning (sec)	Word Ending (sec)	Peak Time (sec)	Peak Value (amplitude)
Photo 1	0.22	0.53	0.4	2.0
Photo 2	0.53	0.75	0.6	4.0
Audio 1	0.4	0.7	0.6	0.84
Audio 2	0.9	0.98	0.94	0.16
Audio 3	0.98	1.18	1.05	0.13

Next, the results of each step the word verification algorithm will be listed.

1. Photo 2 would be matched to Audio 1. This would have been the only match since all other photo-to-audio peak distances are greater than 0.2 seconds

2. Since it has the maximum peak amplitude (0.84), Audio 1 would be selected as the initial audio word. Its endpoints (0.4 and 0.7 seconds) would be selected as the correct word boundaries.

3. Photo 2, which was matched in step 1 to Audio 1, would then be selected as the initial photo word. Its endpoints (0.53 and 0.75 seconds) would be selected as the correct word boundaries.

4. No other audio word peak is within +/- 0.25 seconds of Audio 1's peak time (0.6 seconds). If, for example Audio 2's peak time was within 0.6 to 0.85 seconds then its ending time would have been selected as the new ending time for Audio 1.

5. Since there was no audio endpoint adjustment made in step 4, there will be no photo endpoint adjustment either.

6. Photo 2's endpoints are now compared to Audio 2's endpoints. Since Photo 2's beginning endpoint is greater than Audio's beginning endpoint minus 0.1 seconds (i.e., 0.3 seconds), no photo beginning endpoint adjustment is required. Likewise, since Photo 2's

ending endpoint is less than Audio's ending endpoint plus 0.1 seconds (i.e., 0.8 seconds), no photo ending endpoint adjustment is required.

The result of the post-processing algorithm would have chosen Photo 2 and Audio 1 as the correct words.

4.2.3 Post processing word boundary verification algorithms used for continuous word recognition using sensor fusion information: Basically there was no post-processing conducted on the gradient search results for continuous word recognition using sensor fusion information. Put simply, the photo segmentation results returned from the gradient search algorithm was the only word boundary information used for both audio and photo signals in conducting continuous word recognition. This clearly discarded word boundary information from the microphone signal and information obtained from the time relationship of the photo signal with respect to the audio signal as depicted in Figure 22. Notice that the first two photo humps are normal and can be directly applied in finding the first two audio words "2-8." However, the second two photo humps are "8s" only. Now observe the position of these two photo humps with respect to the corresponding audio humps - the photo humps lag the audio humps. This time/phase relationship, not used in this thesis, can be used to "ferret-out" hidden microphone words.

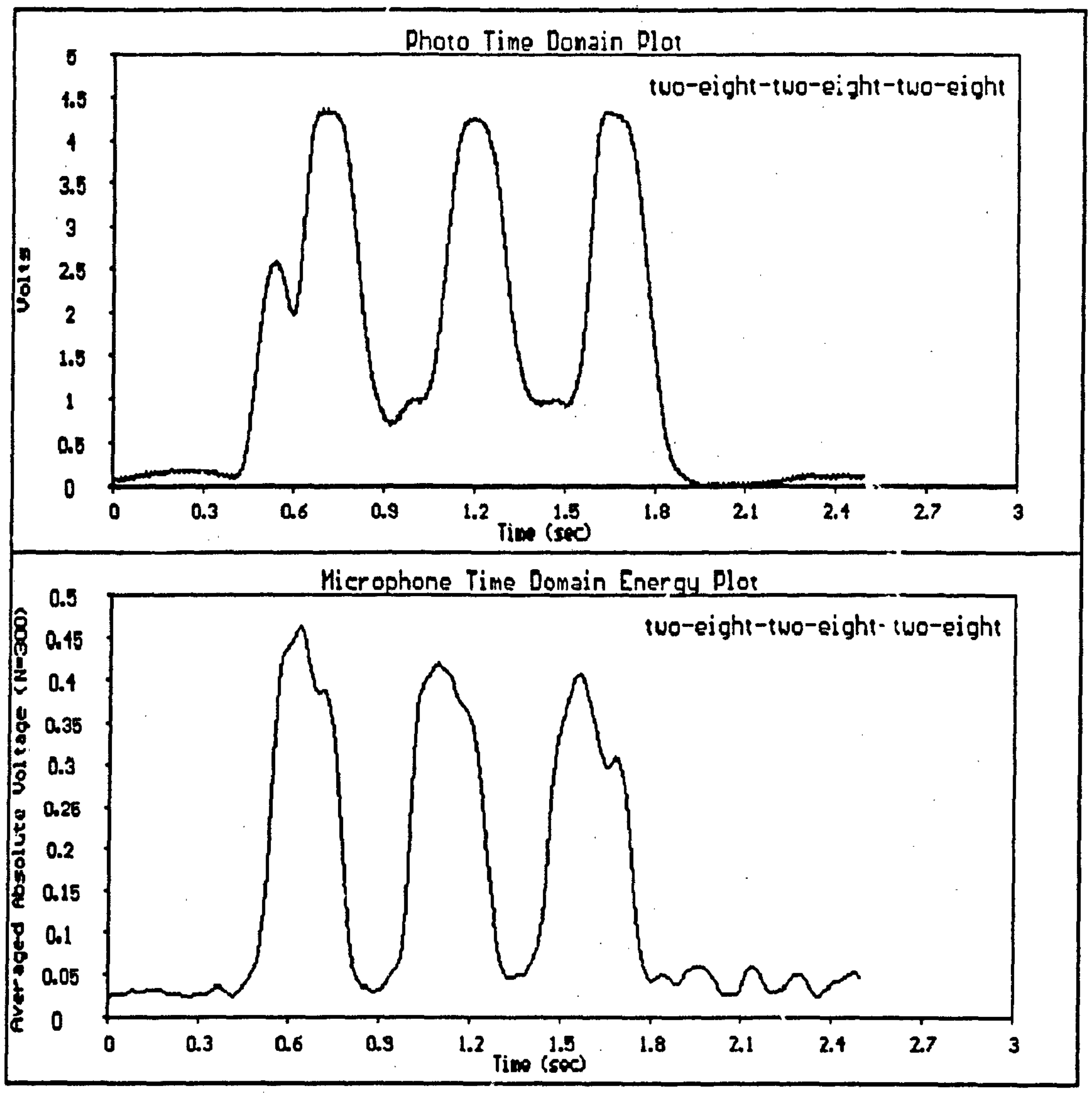


Figure 22. Example of comparing the phase of the photo and audio signals

4.3 Word Segmentation: After the word was found and processed, another algorithm would then strip the word(s) out of the speech time sample given the beginning and ending word boundary times.

4.3.1 Photo segmentation post processing: Since the photo word was sampled at such a high rate (2.5 Ksamples/sec), there were about 2500 float values obtained for every second of photo data. This was 500 times the Nyquist rate of 50 samples/sec. Because of this, it was decided that the number of samples used would be decreased from 2500 to around 100 for a one second sample. This saved processing time while keeping the integrity of the signal intact since the resultant photo word was, in effect, sampled at 100 samples/sec. This was still twice the Nyquist rate and sufficient to avoid aliasing. The last step was to normalize the word(s) with respect to its energy content.

4.3.2 Audio segmentation post processing: The stripped-out audio signal was transformed into the frequency domain using a DFT. Filter banks were then used to process the resultant signal in order to reduce computation time. Several different filter bank processing schemes were tried for the purpose of increasing audio recognition rates. The best results were achieved using the current algorithm which is: between 0 and 2 KHz, approximately 167 12 Hz bandwidth filter banks were used and for the 2 KHz to 4 KHz interval, approximately 33 60 Hz bandwidth filter banks were used. The log of the signal's average amplitude was then computed and the result was placed in one of 200 bins. Lastly, the logarithmic filter bank results were normalized with respect to their energy content.

4.4 Template Processing: Because the mouth moved at a maximum rate of about 25 Hz, it was decided that the photo signals in the time domain would be used as the main feature to perform photo ASR. This decision was based on the fact that in the frequency domain, there is very little photo information below 25 Hz that can be utilized to perform ASR experiments. In fact, some experimentation has revealed that most of the photo energy content for most words lies below about 10 Hz of which an example is shown in Figure

23. As a result, photo template processing and speech recognition was

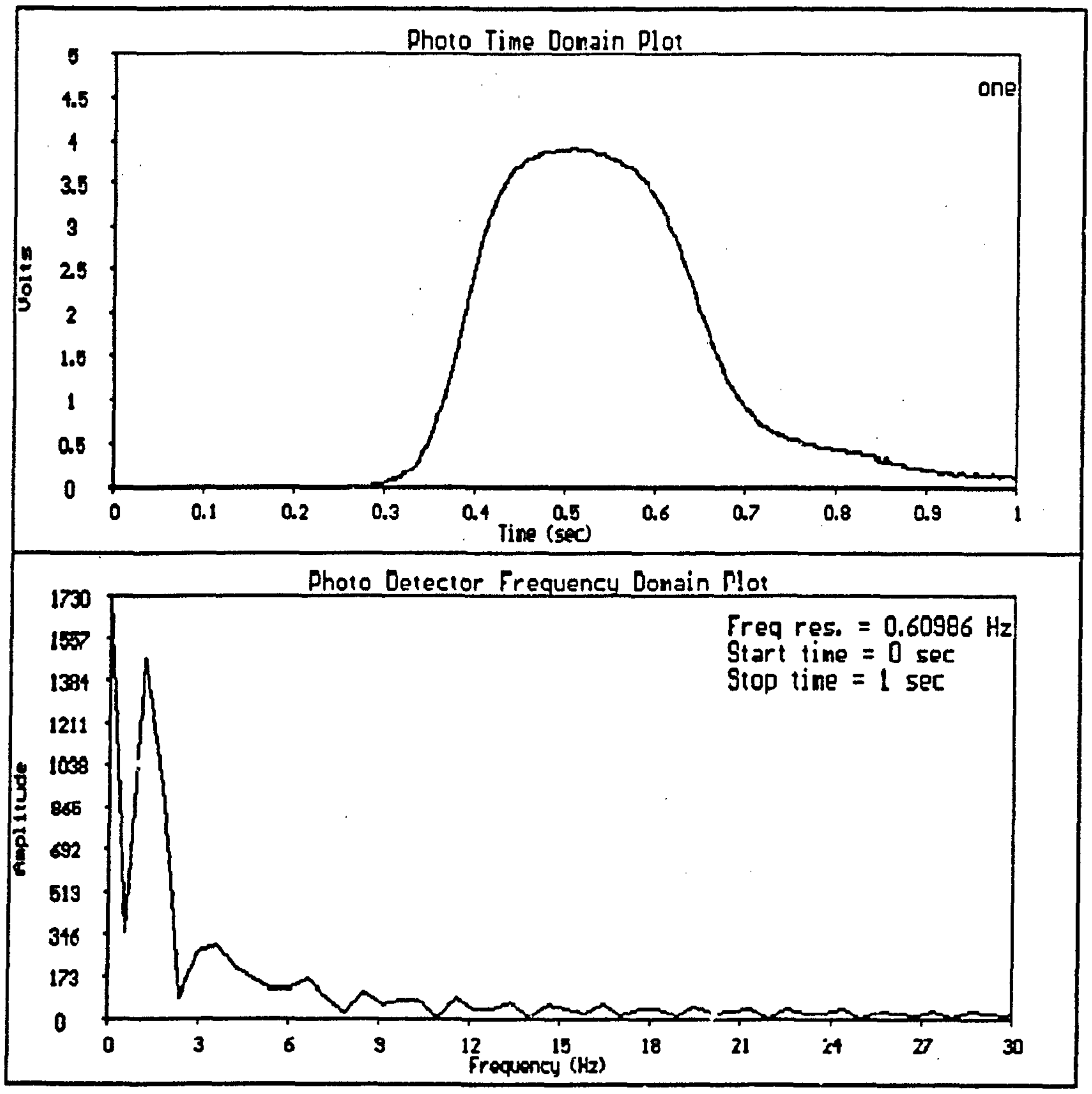


Figure 23. Example of photo frequency information content for the word "one"

accomplished in the time domain. However, since the cutoff for audio was 5 KHz, there was much more spectrum information available in the audio frequency domain as compared to the photo frequency domain. This can be verified by comparing the audio's frequency spectrum of Figure 24 to the photo's frequency spectrum in Figure 23 for the same word and speaker. Consequently the frequency domain was chosen for the audio signal to perform template processing and eventually speech recognition.

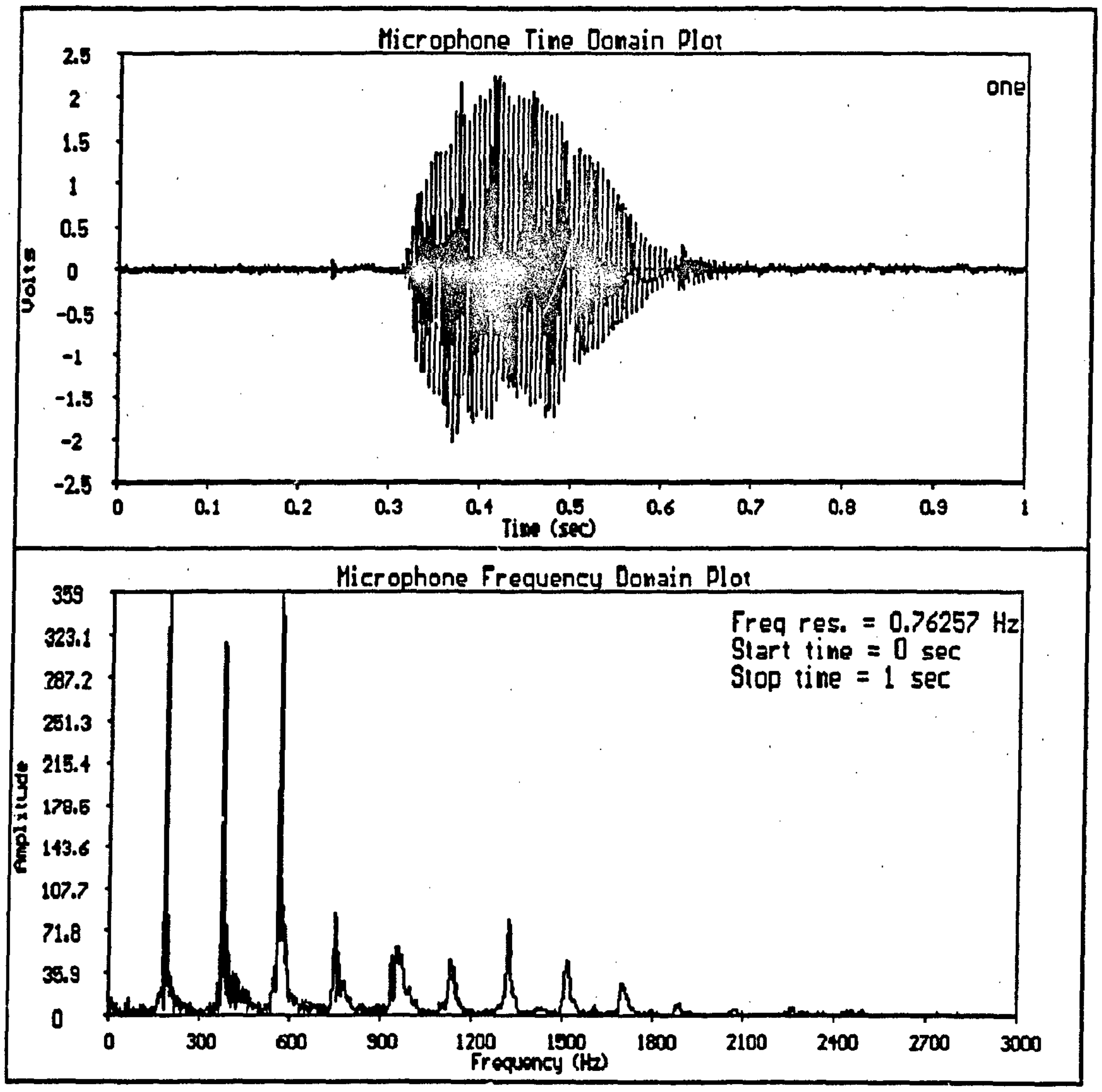


Figure 24. Example of audio frequency content for the word "one"

Two type of templates were created: (1) Speaker-dependent and (2) Speaker-independent templates. A Dynamic Time Warping (DTW) function was used to select which one out of the five word multiples, for each word, for each person was to become the speaker-dependent template. This algorithm compressed or stretched a waveform with respect to a reference template and returned the cost to obtain the optimum path. Each multiple of a word was, in turn, a reference template that was compared to the other four word multiples. The reference template with the overall lowest cost as compared to the other four word multiples became the template for that particular word.

The DTW algorithm was also used to process speaker-independent templates. The results of the speaker-dependent templates were used to decide which template would become the "speaker-independent" template. Each speaker-dependent template for each isolated word, for each person was, in turn, a reference template that was compared to another person's speaker-dependent template. The reference template with the overall lowest cost as compared to the other four speaker-dependent templates became the "Speaker-independent" template for that particular word. Examples of an actual photo and audio templates can be seen in Figures 25 and 26 respectively. Note that all templates were extracted from "isolated" words. No templates were derived from continuous speech in this effort.

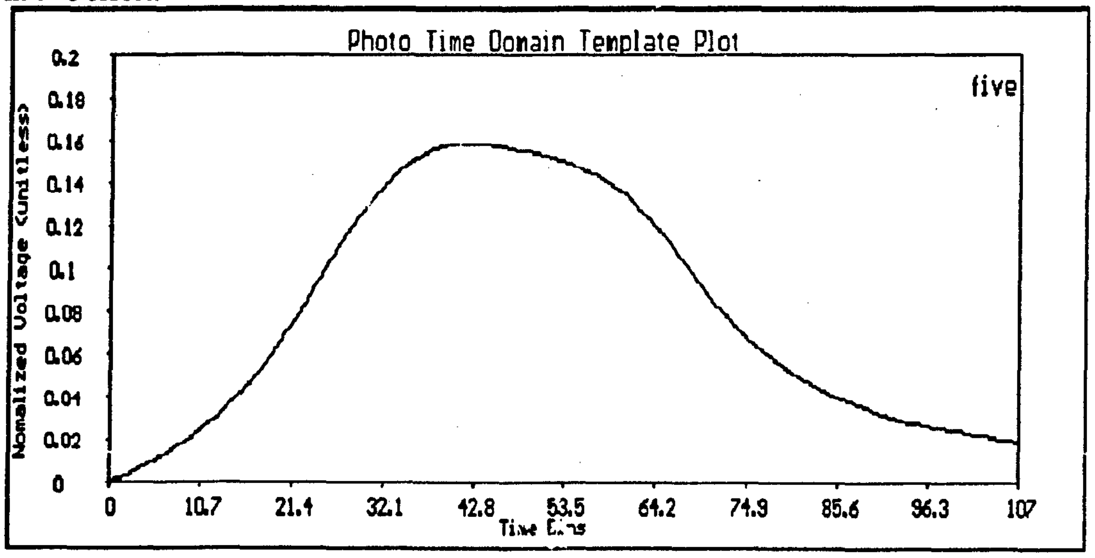


Figure 25. Example of a photo template for the word "five"

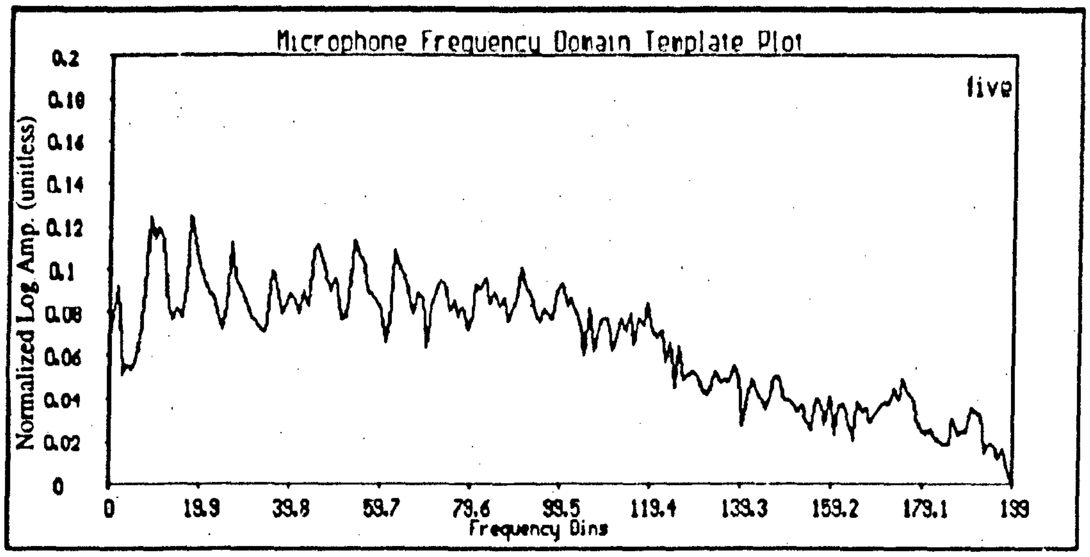


Figure 26. Example of an audio template for the word "five"

4.5 Speech Recognition:

4.5.1 Non-sensor fusion word recognition: Each person's isolated and continuous words from Table 1 were compared to that particular person's speaker-dependent template and the speaker-independent template. The photo and audio templates with the overall lowest DTW scores were chosen as the recognized word.

4.5.2 Sensor fusion word recognition: This part of the fusion process was identical to the standard definition of sensor fusion, "Two different systems, each of which produces an output word guess independent of each other, are fused together to obtain a best guess." The photo and audio templates with the overall lowest DTW scores were compared using the sensor fusion decision theory described below to decide which word was the best guess.

- The photo template was selected if the audio DTW cost was greater than 0.17 and the photo DTW cost was less than 0.005.
- The photo template was selected if the audio DTW cost was greater than 0.6 or the photo DTW cost was less than 0.0001.
- Otherwise the microphone template was chosen.

These were arrived at empirically with the objective of achieving as high a recognition rate as possible.

V. Test Results

The speaker-dependent experiments used each person's "speaker-dependent" template (i.e., "0" through "9" and "yes" and "no") to compare that particular person's utterance for the words in Table 1. There were five utterances of each word type so the results are a compilation of 5 utterances of 12 isolated words and 5 utterances of 5 continuous words.

The speaker-independent experiments used the "speaker-independent" template (i.e., "0" through "9" and "yes" and "no") to compare that particular person's utterance for the words in Table 1. Again, there were five utterances of each word type so the results for the speaker-independent experiments are a compilation of 5 utterances of 12 isolated words and 5 utterances of 5 continuous words.

The results for six experiments are shown in Table 3 and a discussion of each result will follow.

Table 3. ASR results

Experiment	Pat	Janet	Don	Mary	Jill
1. Isolated photo word, speaker-dependent recognition using no-sensor fusion information.	62%	55%	52%	35%	12%
1. Isolated audio word, speaker-dependent recognition using no-sensor fusion information.	87%	88%	87%	88%	53%
2. Isolated photo word, speaker-independent recognition using no-sensor fusion information.	22%	42%	15%	12%	7%
2. Isolated audio word, speaker-independent recognition using no-sensor fusion information.	33%	37%	40%	40%	17%
3. Isolated word, speaker-dependent recognition using sensor fusion information.	90%	92%	87%	93%	63%
4. Isolated word, speaker-independent recognition using sensor fusion information.	38%	42%	42%	42%	20%
5. Continuous photo word, speaker-dependent recognition using no-sensor fusion information.	10%	20%	19%	17%	5%
5. Continuous audio word, speaker-dependent recognition using no-sensor fusion information.	42%	27%	46%	40%	30%
6. Continuous word, speaker-dependent recognition using sensor fusion information.	40%	29%	36%	32%	30%

5.1 Isolated word, speaker-dependent recognition using no sensor fusion information: As expected the audio signal performed better than the photo signal. However 95% to 100% audio isolated word recognition rates are achievable today for many ASR systems. So it was disappointing that the audio did not perform better. This was especially important for the continuous word recognition experiments using sensor fusion where the photo information was used to find the audio words. This is useless if audio's recognition algorithm is not very good. It was interesting to note that the word "one" did poorly for all subjects.

The worst audio results, "Jill", were due to the inherent design of the speech processing algorithms. Recall that the gradient-search algorithm searched for significant changes in amplitude to decide where word boundaries were located. If it encountered more than one isolated word then the algorithm would return multiple words even though there was only one real word. Unfortunately quite a few of Jill's isolated audio word signals contained one or more small "humps" after the actual word was spoken. These are natural breath sounds and an example can be seen in Figure 27.

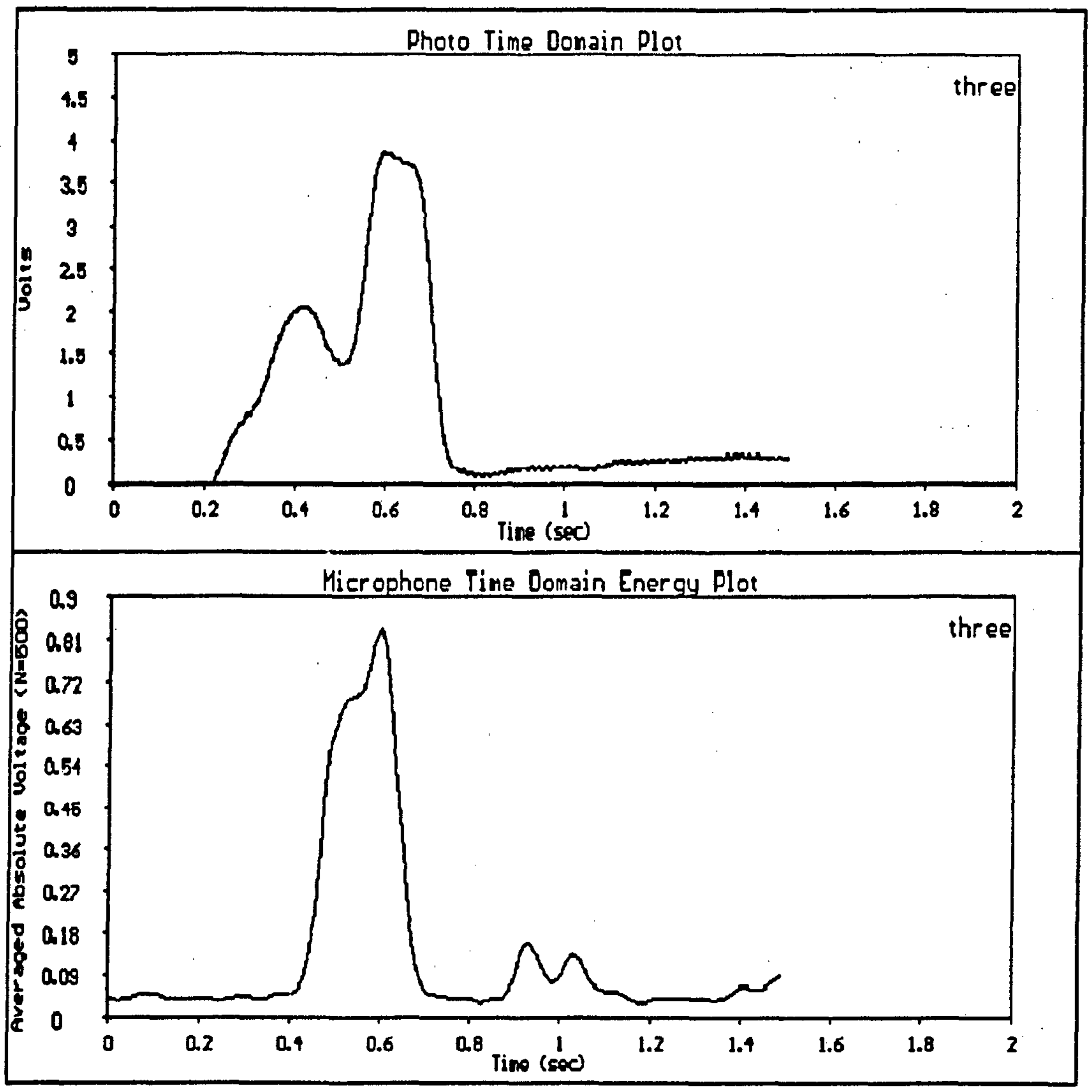


Figure 27. Example of Jill's audio word "three" with breath noise

The results for the photo signal recognition were good. The second worst photo results, "Mary", were due to the fact that this person simply did not keep her mouth from opening or moving prior to saying the actual word. An example of this can be seen in Figure 28. This is quite natural but played havoc with the algorithm trying to find the

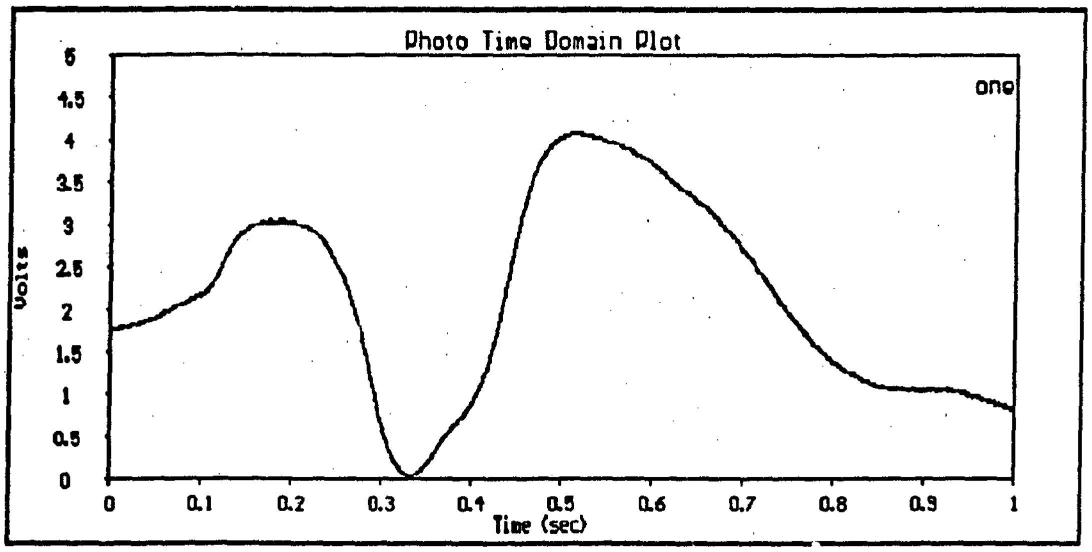


Figure 28. Example of "Mary" not keeping her mouth shut prior to speech

words.

"Jill" also had the worst photo results. There were problems with the headset and offset adjustments during data recording. This caused some signals to hit the rails and become basically useless for speech recognition. New data should have been recorded for Jill which would have certainly improved her photo results.

5.2 Isolated, speaker-independent recognition using no sensor fusion information: The recognition rates for both the audio and photo information is fairly poor. This is understandable for the photo signal since there were clear differences between the test subjects as shown in Figure 29. However, the audio should have performed better.

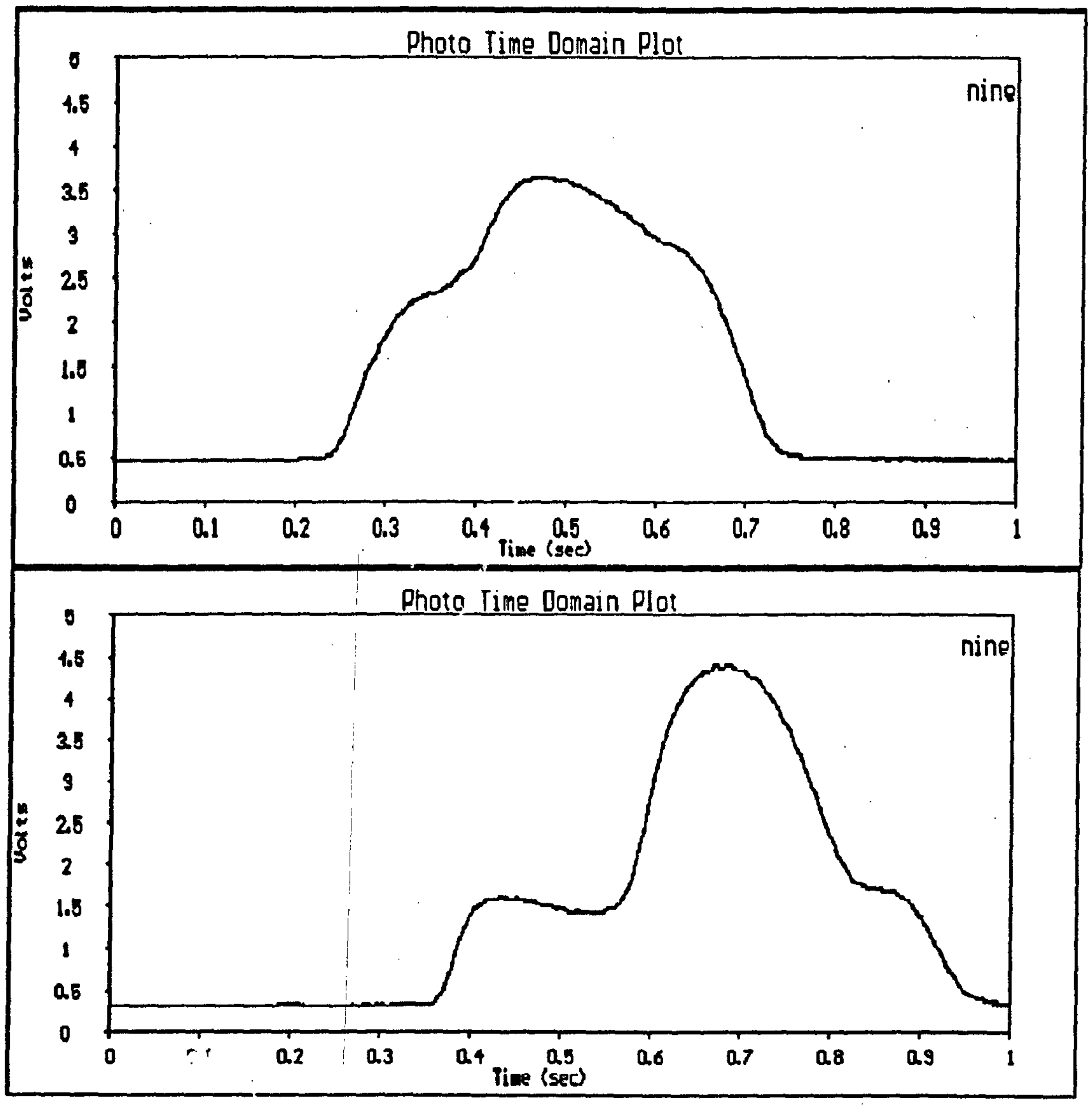


Figure 29. Example of photo differences between subjects

5.3 Isolated word, speaker-dependent recognition using sensor fusion information: One has to compare these results with the microphone results for experiment 1. This comparison clearly shows that the objective of this thesis has been met - "Using additional information obtained from IR or visible detectors does increase the recognition rate of an audio ASR system." About 10% of the words were chosen from the photo results. The rest of its success is due to using the microphone to help verify photo words. These results could possibly be improved. As we saw earlier in Figure 27, some subjects had breath noise. In these cases it might have been better to use the photo words to locate the word boundaries of both signals.

5.4 Isolated word, speaker-independent recognition using sensor fusion information: One has to compare these results with the microphone results for experiment 2. Again these results clearly show that using additional photo information can increase recognition rates.

5.5 Continuous word, speaker-dependent recognition using no sensor fusion information: The photo ASR system performed fairly poor. The main problem in using photo words in the continuos domain is that the words are usually merged together. Using templates that were derived from isolated words correlated to continuous words only if there was a clear separation between the words. Word separation depended on the position of the words with respect to each other.

The microphone results were not much better. Most of the problem was finding the words. This is understandable since it is the crux of continuos ASR. However, the other main problem was that the recognition algorithm was performing so poorly.

5.6 Continuous word, speaker-dependent recognition using sensor fusion information: One has to compare these results with the microphone results for experiment 5. The results are not that terrible considering the fact that only photo word boundaries were used to locate words in both the photo and microphone continuous signals. Note that most of the words were picked from the microphone results. Put simply, isolated photo templates perform poorly when used to perform continuous word ASR. This was observed in experiment 5 where the photo percentage rates were very low.

Note that there are two experiments missing: "5.7 Continuous word, speaker-independent recognition using no sensor fusion information" and "5.8 Continuous word, speaker-independent recognition using sensor fusion information." Since the results for experiments 5.5 and 5.6 were so poor, these last two experiments using cross-speaker recognition would be worse and would probably be useless. As a result, they were eliminated as experiments.

VI. Conclusions and Recommendations

This thesis has clearly proven that using additional information from photo detectors can increase recognition rates. These results are significant considering no optical focusing elements were used for the detectors. Unfortunately one has to put a time limit on such R&D efforts and a lot of my time was spent trying to come up with good photo signals and sometimes microphone signals, and fighting with software bugs. This work is by no means finished. Additional experimentation is required for the continuous sensor-fusion case to clearly show that the results will be superior to using microphone ASR alone. I am fully confident that this can be done. What I have chosen to do for this thesis was the simplest continuous sensor-fusion case and is not the optimum case. There is additional information that can be gained through the comparison of both signals as we saw in Figure 22. I chose to discard this extra information to make the programming easier and the time to finish shorter.

My recommendations for future microphone/photo efforts are listed below.

- Use the current sensor fusion algorithm for isolated ASR. As we saw, using the microphone signal to verify photo words and then comparing the recognition results of each signal can result in a superior isolated ASR system.

- For the continuous ASR system, some investigation and experimentation should be accomplished to obtain more information on the location of the microphone words. In this case, use the photo signal to mainly help locate the microphone words. This can be accomplished easily when there are dips and pauses in the photo signal. Something that I did not do was to use the magnitudes of either signal (photo or microphone) to determine if a valid word boundary point was found. In addition, I recommend using the time/phase relationship information of the photo words with respect to the microphone words to help locate microphone words (see Figure 22). I recommend not using the photo recognition results. Unless there is a clear separation (i.e., a pause) between words, photo continuous ASR performs poorly. The results are probably not worth the additional processing time.

- A better headset should be designed that is lighter and more ergonomic. Modern microphone headsets use lightweight plastics to achieve these requirements. However, this problem could naturally go away for some pilots who have their helmets specially fitted. Photodiodes and a LED could then be easily mounted into the helmet.

- An optical attachment to the photo detector such as a Fresnel lens or a miniature convex thin lens should be experimented with. No adjustment of the photo detector should be required after the headset (or helmet) is worn. To decrease ambient light noise, a spatial filter should be used in addition to the Fresnel lens or miniature thin lenses. However, this may not be required since some lenses are also filters.

- The "h(t)" headset movement noise term needs to be eliminated. Maybe the headset improvements suggested above will help decrease this noise. If not , addiiional signal processing (hardware and/or software) could be used to try and eliminate "h(t)". One way would be to use the second photodiode to subtract out headset movement. The setup would use the one detector to measure "h(t)" only and then subtract it out from the other detector's signal in the analog circuit by using a differential amplifier.

- Using filter banks for the audio ASR process is probably why my audio recognition rates were so low. I recommend adding other recognition schemes (e.g., zero-crossings, formants, etc.) which are probably better that the single one I have chosen for the microphone.

- I recommend that if the CW/AM system is used, try optical fibers to send the light to a focusing lens. Then use another fiber with another focusing lens to receive the reflected photons.

- I used the microphone signal to help verify the photo signal for both template processing and ASR using sensor fusion. If these same experiments were performed under real-world conditions such as an aircraft cockpit, the current algorithms may not operate properly. Consequently one may in this case rely more heavily on the photo signal to help locate both photo and microphone words.

- IR ASR certainly deserves some additional attention. The ideal application may be continuous ASR where the location of the IR humps could be used mainly to help locate the audio words.

- An additional area that photo speech technology may be applied to is speech therapy. This idea came to me when I noticed that I was saying the photo word "zero" incorrectly for which I promptly corrected. However it was harder for me to tell that I was slurring the word "zero" using only its audio energy distribution.

- Using a PC with MS-DOS has its shortcomings. I had to implement dynamic memory allocation algorithms to circumvent memory limitations. The worst was the DTW algorithm. Using conventional memory allowed only for a maximum of a 320 X 320 float element array. Fortunately about a 100 X 100 size array was only required for this thesis. I suggest using either a mainframe computer or the newest version of

Microsoft C++ for the PC which allows access to extended memory.

- Lastly, here are some recommendations for the A/D board. Try using a First-In-First-Out (FIFO) chip in the digital board to take some burden off of the PC interface circuitry. Also, try using one DMA channel to transfer the photo signal and the other to transfer the microphone signal. This will automatically decouple the two signals if using one A/D chip. However, this method will transfer data at a slower rate but should suffice if the sampling rates are low.

Appendix A. Data Acquisition Board Design

A system block diagram of the digital data acquisition board can be seen in Figure 30. This was a two-channel board that plugged into a standard IBM PC. One channel was

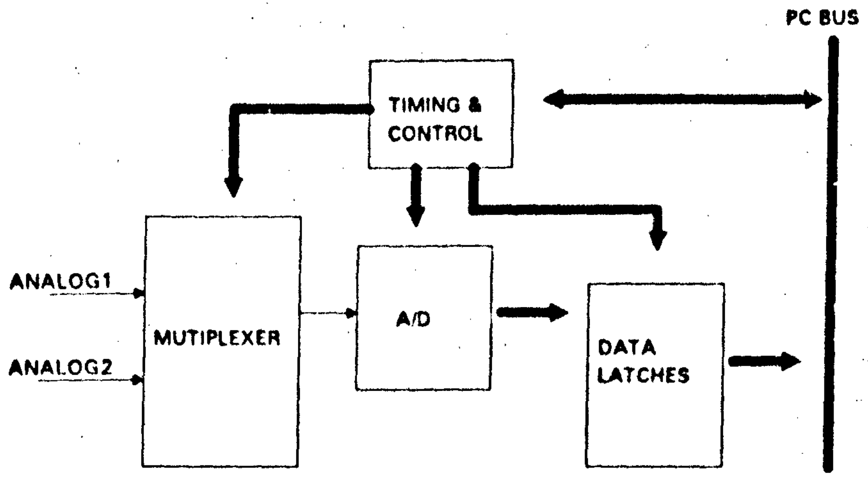


Figure 30. Block diagram of A/D board

used for the A/D photo signal and the other for the microphone signal. The signals were time-multiplexed in order to share the same 12-bit A/D chip. After conversion, the lower byte and upper nibble of each signal were temporarily stored in latches awaiting acknowledgment from the computer. The most complex section of the block diagram is the timing and control. It directed to the multiplexer when it was time to switch input signals and initiate A/D conversions. It also had to handshake with the PC's microprocessor and DMA (Direct Memory Access) chip(s) to download digitized data to the hard drive.

The microphone signal was sampled at 25 KHz and the photo signal at 2.5 KHz. Therefore, the current lowpass filters (5 KHz for the microphone and 25 Hz for the photo) were adequate to avoid aliasing.

The design specifications called for using 8-bit data transfers for the case where this board was to be used in a XT PC. This definitely slowed the data transfer rate as compared to a 16-bit bus. But the advantage was that this board would be compatible in any standard PC. Incidentally this board started in a XT PC and ended up being used in a 386 PC.

Due to the audio's high sampling frequency (25 KHz), DMA channels were used to transfer the data from the digital board onto the computer's hard drive. This was faster than programmed I/O but unfortunately the interface circuitry was more complex.

The interface circuitry that performed the handshaking with the 8237-5 DMA controller chip is shown in Figure 31. This is one of three sheets that made up the A/D board. This circuit was controlled by both the DMA chip and computer software. Two DMA channels (1 and 3) were used to increase data transfer rates to computer memory. When data was ready to be transferred (i.e., it's been digitized), a DRQ (direct memory request) line was raised from the PC board. The DMA controller then prioritized the PC board's request and eventually responded by lowering the corresponding DACK (direct memory access acknowledge) line. The PC board then used this signal to reset the DRQ line back low.

When the IOR (I/O read) line went low, the address on the bus was valid and the PC board responded by putting its data on the bus. Eventually the DMA controller reached its preprogrammed number of transfer cycles and raised the T/C (terminal count) line. The A/D board sensed this condition and responded by raising the IRQ3 line which told the software when data in memory was ready to be downloaded to the hard drive. When the data transfer was complete, the software would then send the "RESET" command to the A/D board which released the IRQ3 line and started the whole process over again.

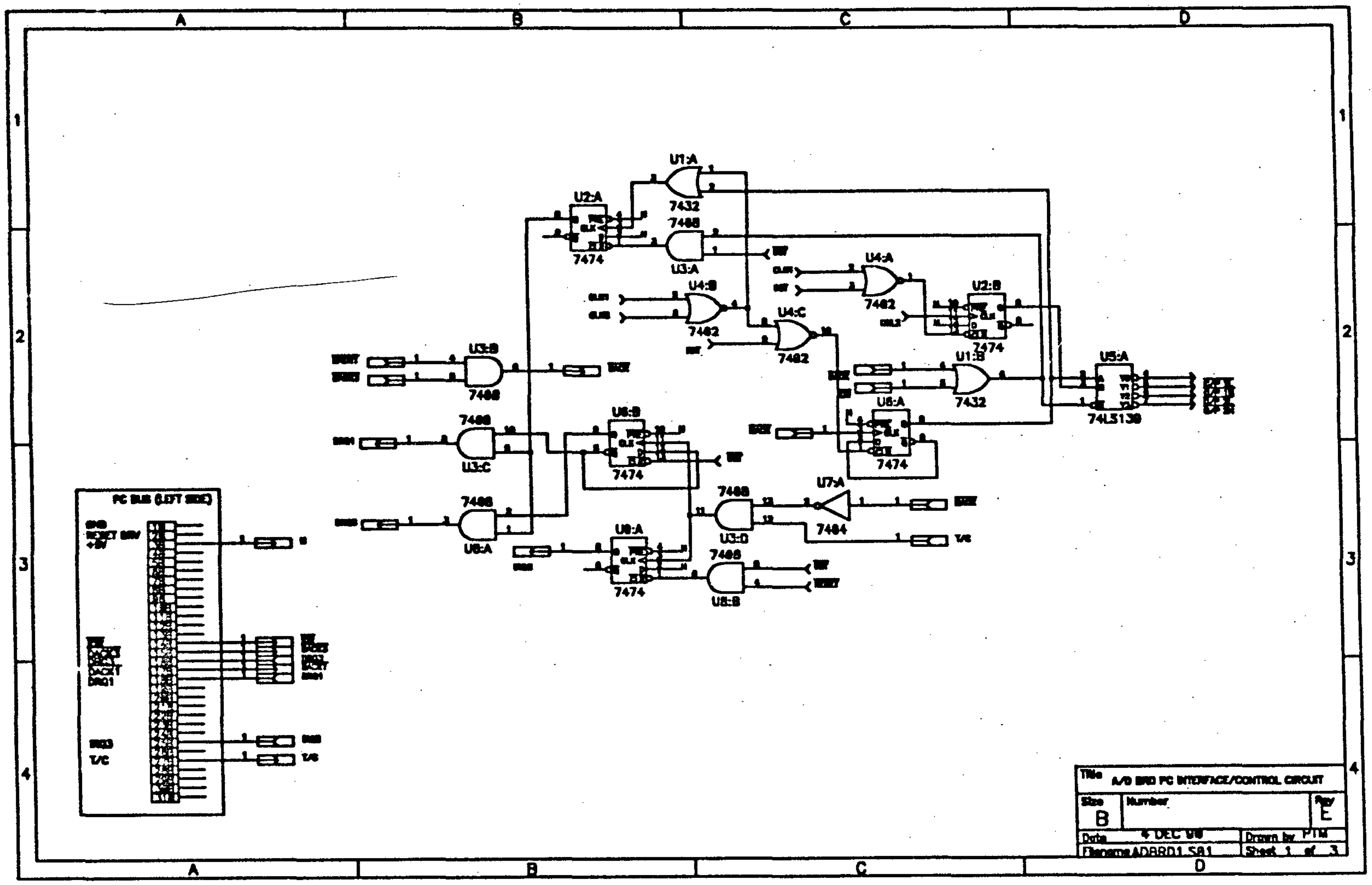


Figure 31. Main PC interface/control circuit diagram (sheet 1 of 3)

The data acquisition process just described is outlined on the next page for a typical DMA sequence [13:114,157-159].

1. The data acquisition board sets either the DRQ1 or DRQ3 signals high when data is ready to be sent. Two DMA channels were used to speed up data transfers.

2. When ready, the 8237-5 chip responds to the request by setting DACK1 or DACK3 low. The board puts the data onto the data bus and drops the DRQ1 or DRQ3 lines.

3. The DMA chip then puts the memory location onto the address bus and reads the data from the board to memory.

4. At the end of a programmed number of bytes transferred, the DMA controller raises the T/C line.

5. The board ANDS these with the appropriate DACK line and raises the IRQ3.

6. The IRQ3 interrupt is then detected by the computer software for the purpose of reprogramming the DMA controller and sending a "RESET" I/O address to the data acquisition A/D board.

7. The board receives the "RESET" command and lowers the IRQ3 line.

8. Steps 1 through 7 are repeated under software direction.

The A/D board's timing diagram is shown in Figure 32. The timing circuit is shown in Figure 33 and the data processing circuit is shown in Figure 34.

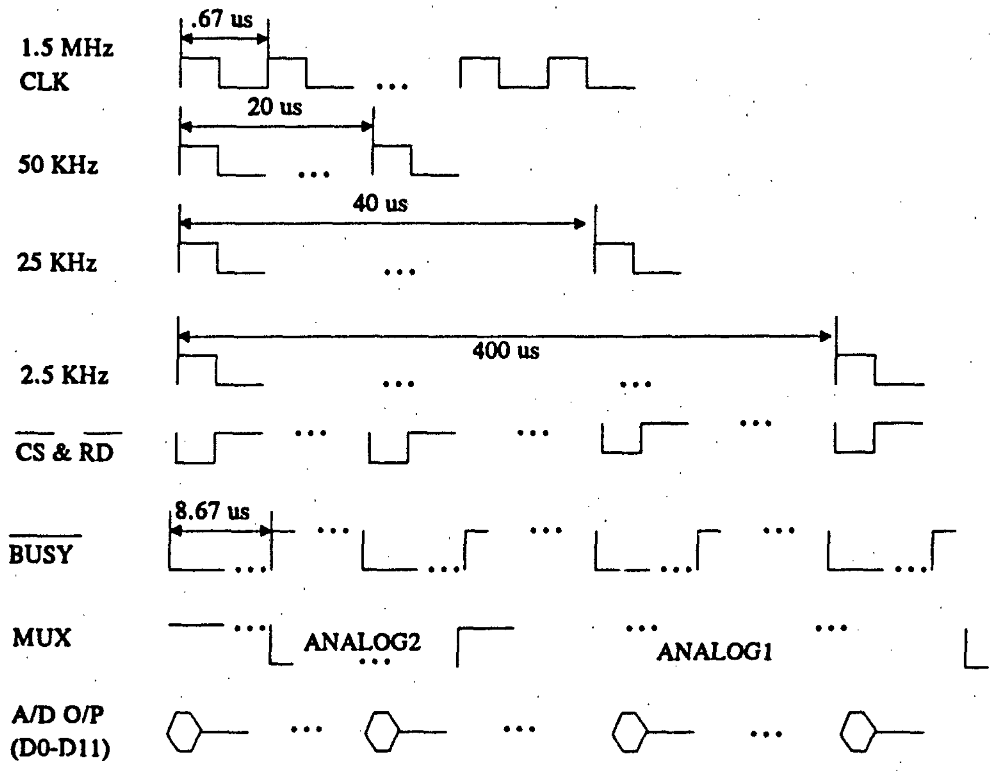


Figure 32. Timing diagram for A/D board

The heart of the timing is a 1.5 MHz clock oscillator which was down converted to three main frequencies: 50 KHz, 25 KHz, and 2.5 KHz. The 25 KHz and 2.5 KHz were the sampling frequencies and the 50 KHz frequency was used to both convert photo signals (ANALOG2) and latch previously converted microphone data (ANALOG1) out to 74374 registers.

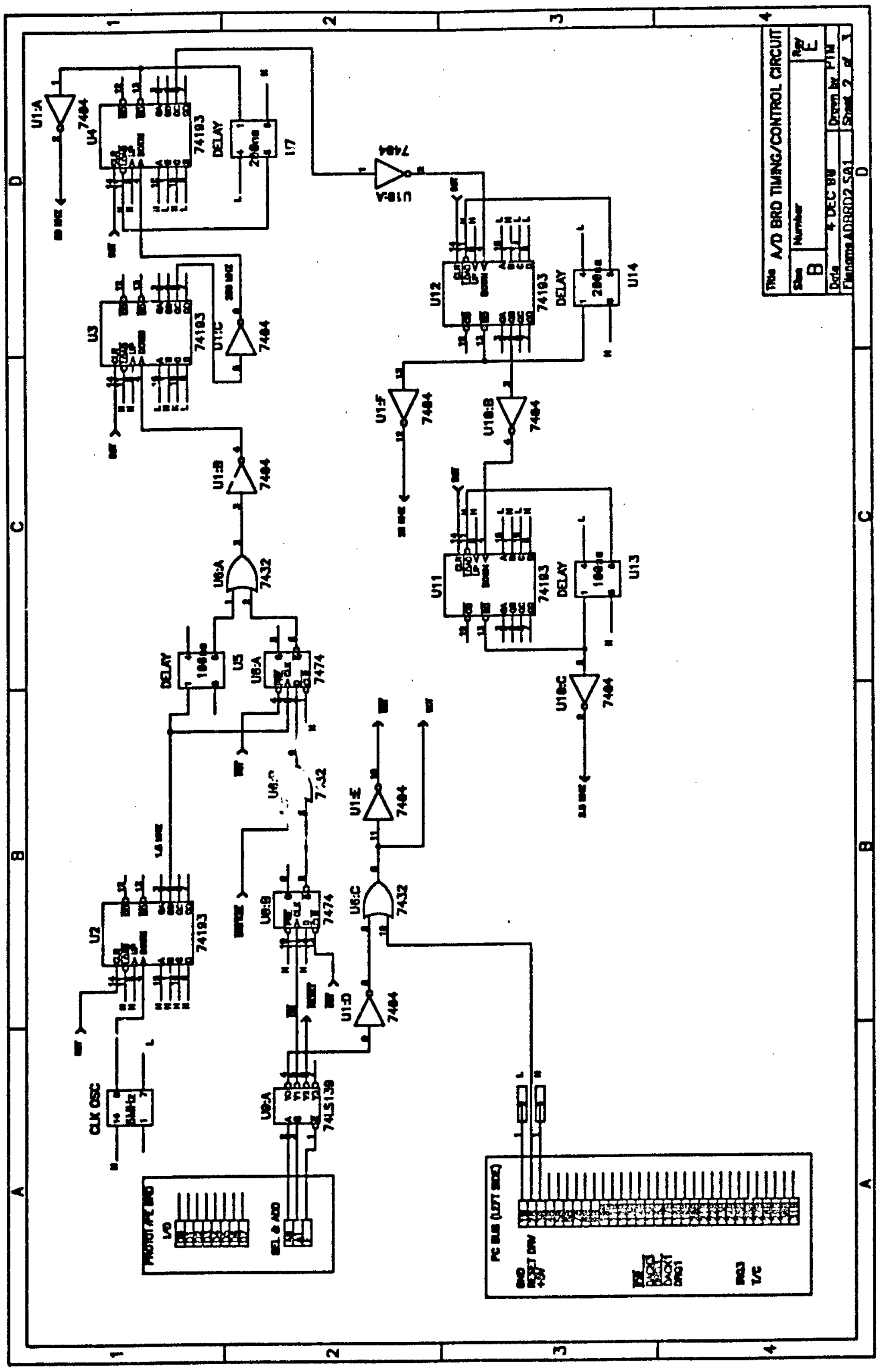


Figure 33. A/D board timing/control circuit diagram (sheet 2 of 3)

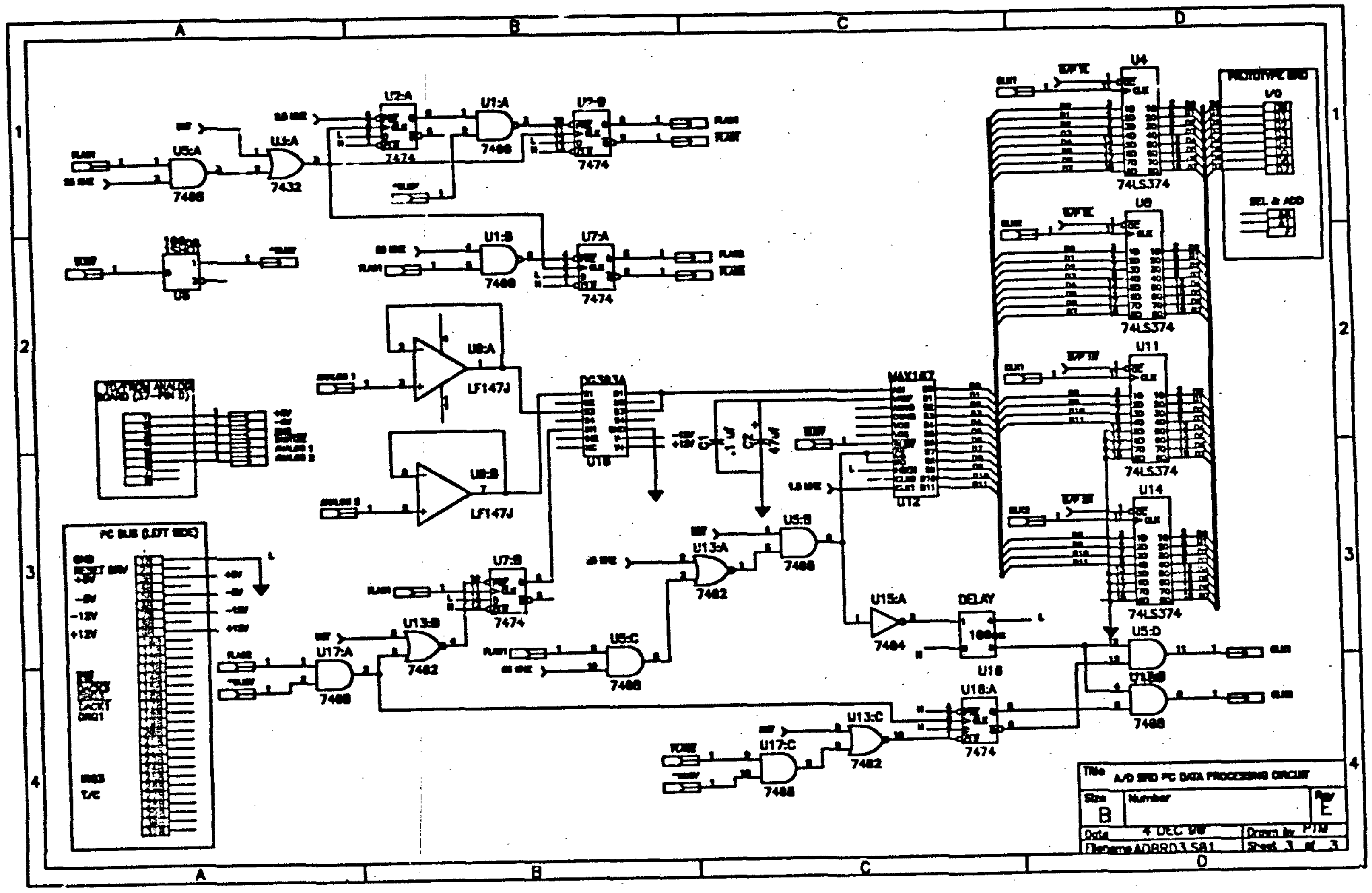


Figure 34. A/D board signal processing circuit diagram (sheet 3 of 3)

The 12-bit A/D chip, a MAX167, starts a conversion when both CS and RD went low. The BUSY signal then went low during the conversion time (about 8.67 usec) and previously converted data appeared at the chip's output (D1/D12) [14]. Normal operation was when the 2.5 KHz pulse was not present. The microphone signal was then continuously converted and read out to two latches at 25 KHz. The lower byte (D7/D0) was stored in one latch and the upper nibble (D11/D8) was stored in the other latch. This continues until the next 2.5 KHz pulse appeared. At this time the multiplexer would then switch inputs and the photo signal was converted while the last microphone conversion bits were read out to latches. This occurred after the 2.5 KHz pulse and at the next 50 KHz pulse. The end result was that for every digitized photo word that is sent, 10 microphone words were sent within the same period.

All three circuit diagrams were built on a single 16-bit prototype card from JDR Microdevices. The card came with built-in I/O addressing already decoded which was convenient since the addresses 300, 301, and 302 were used to initialize the PC board, activate the timing circuit, and reset the IRQ3 interrupt respectively. A 74139 chip can be seen decoding these addresses in Figure 33. As shown in the schematic, the "ON" command was used to start the timing after the board was initialized using the "INIT" command. The "RESET" command, which was generated but not used in Figure 34, was used in Figure 31 to reset the hard interrupt.

Appendix B. Software Design

The comp:ler and language used for this project was Borland C++ for a PC. A system block diagram of the software is shown in Figure 35. All software source code is included in Appendix C.

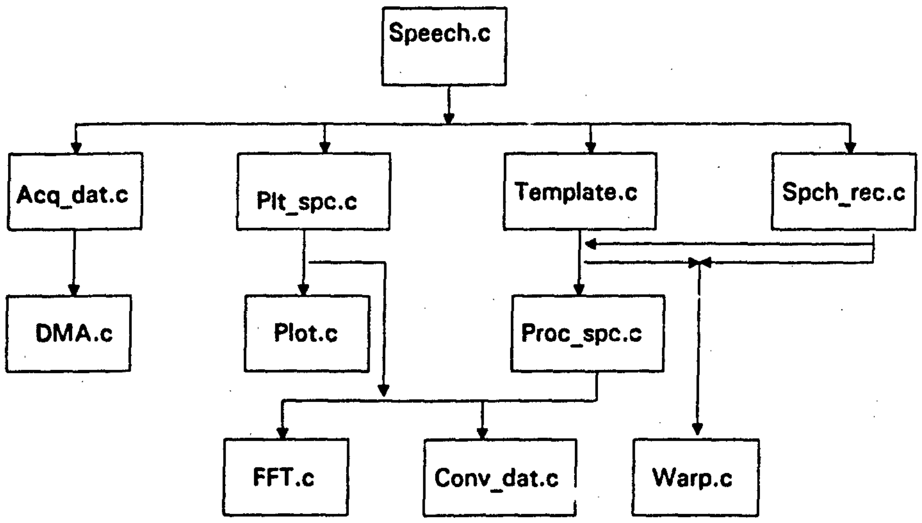


Figure 35. Software program block diagram

As can be seen in the block diagram, there are four main sub-programs: Acq_dat.c, Plot_spch.c, Template.c, and Spch_rec.c. These were controlled by the window-driven "Speech.c" module which received user-inputted information. The four main sub-programs will now be discussed.

Data Acquisition: To conduct DMA transfers, the 8237-5 chip had to be re-programmed before the board was allowed to transfer data. The chosen modes of operation were as follows:

- Read to memory operations only
- Single byte transfers

The memory address for start of transfer and the byte count was also programmed into the DMA chip.

The data acquisition software programs are "Acq_dat.c" and "DMA.c". These performed all of the programming, handshaking, and data transfers with the 8237-5 DMA controller and digital board. "Acq_dat.c" was the main program that controlled the data transfers. It used miscellaneous DMA routines contained in "DMA.c" most of which were found in an excellent article on DMA interface hardware and software design [15].

When called, "Acq_dat.c" initialized and activated the acquisition board through I/O addressing. It then created two buffers using dynamic memory allocation to conserve memory. One buffer was used to record the data through DMA transfers while the other, already filled buffer was being written to disk. Even though these operations appeared to be accomplished in parallel, they weren't. The DMA chip still had to steal clock cycles from the CPU to download data to disk.

As mentioned earlier in the digital board design, an interrupt (IRQ3) was used to tell the computer software when a DMA transfer is completed. The software then switched buffers and, using the "RESET" command, reset the board to continue the data acquisition.

The format of the received data was such that one block of data contained 10 integer values of microphone data and 1 integer value of photo data. The number of integer values transferred depended on the time length of the data to be converted using the A/D acquisition board. For example, if one second of data was to be recorded then this equated to (1 sec)*(25000 samples/sec) which is equal to 25000 integer values of microphone data and (1 sec)*(2500 samples/sec) which is equal to 2500 integer values of photo data. Both signals were then stored together in one binary file. When required, the data was read back by the "conv_dat.c" program, demultiplexed, and converted to float values using the A/D 12-bit transfer function.

Data Plots: "Plt_spc.c" plotted both time and frequency data for both signals. To plot amplitude versus frequency, a FFT was performed on the time information through the use of "FFT.c". Frequency resolution was a function of the available memory and the time length of the signal. To setup the graph and plot data, the program "Plot.c" was used. The primary reasons for the data plotting program was to observe the results of algorithms used and record results for this report.

Template Processing: "Template.c" created the templates for both signals. The first step before template processing was to locate the isolated words. Word boundaries and additional information (e.g., peak location, beginning time, ending time, etc.) about the signals were returned by the "find_words()" routine which was called by the "sensor_fusion_segmentor()" routine. The "find_words()" routine, which is in the "Proc_spc.c" program, located both photo and microphone words. It sensed huge changes in amplitude to find the words since the boundaries were somewhat a function of changes in signal levels. It incorporated a gradient search method to locate slopes of sufficient magnitude which were assumed to be word boundaries.

An algorithm called "sensor_fusion_segmentor()" was programmed to decide where the words were located using the results of the find_words() routine. The sensor_fusion_segmentor() routine's sole purpose was to verify and correct if necessary the beginning and ending word boundaries for a given speech signal. The audio signal was used in this routine to verify the photo's boundaries.

After the word boundaries were verified, two routines, "process_photo_data()" and "process_mic_data()," were then used to strip the word out of the speech time sample. They are located in the "Proc_spc.c" program. Their sole purpose was to convert the signal for speech processing and then return only the data within the beginning and ending times sent to them. The audio signal was transformed into the frequency domain in the "process_mic_data()" routine. Both the photo and audio words were then filtered using a smoothing algorithm called "filter()" which is located in the "Proc_spc.c" program [11:257]. This algorithm low-pass filtered the energy distribution to remove unwanted spikes.

The words are normalized with respect to their energy content after being processed by either the "process_mic_data()" or the "process_pho_data()" routines. The normalization routine is called "normalize()" and is located in the Proc_spc.c program.

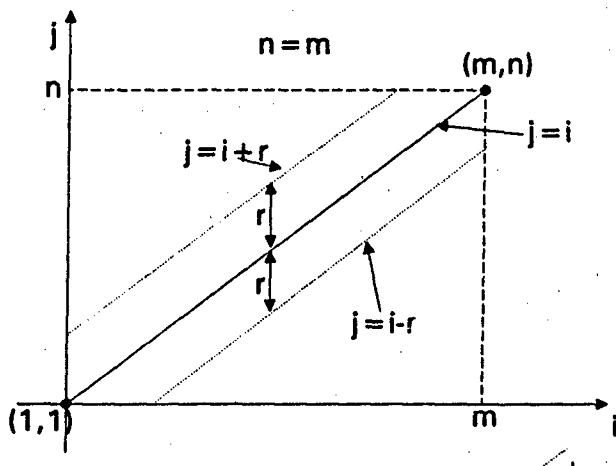
A Dynamic Time Warping (DTW) function was then used in the "Template.c" program to select which word multiple out of five word multiples was to become the word template. The DTW algorithm used extracted from a text book on speech recognition [10:379-382]. The author mentioned that this was a bare-bones version of the algorithm. So some modifications were necessary after converting the routine from FORTRAN to C++.

The DTW's original path width was hard-wired to +/- seven to limit the amount of warping that was allowed. To add some flexibility for optimization, the constant path width was changed to a variable path width for the algorithm.

Another path constraint during the warping process was to allow only one step in either the vertical, diagonal, or horizontal directions. This was changed from one step to a variable number of steps. However, one had to use this change with caution. Too many steps in either the horizontal or vertical directions will result in a distorted waveform that is either stretched or compressed out of proportion.

The DTW algorithm's path width was originally calculated from a straight line that went from (1,1) to (m,n) as shown in Figure 36a. This worked fine if the m = n. When this condition was not met, the results were disastrous as can be seen in Figure 36b. Consequently the algorithm was modified to have a variable slope and offset line to calculate path widths from.

(a.)



(b.)

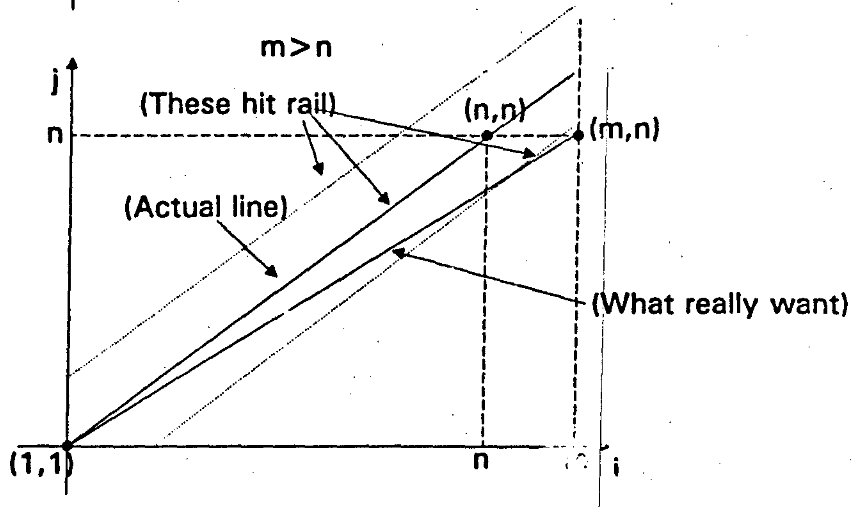


Figure 36. (a.) DTW algorithm path with m=n (b.) DTW algorithm path with m>n

Speech Recognition: The last main program, "Spch_rec.c", performed both isolated and continuous speech recognition. This program used the same operations as "Template.c" but for isolated or continuous word versus templates. The steps for isolated and continuous word recognition using "no" sensor fusion are listed below:

• The word(s) was found using the "find_words()" routine.

• The word(s) was then stripped out and processed using the "process_photo()" and process_mic()" routines.

• The word(s) was then normalized with respect to its energy content using the "normalize()" routine.

• The template(s) with the overall lowest DTW score was chosen as the recognized word.

The steps for isolated and continuous word recognition using sensor fusion are listed below:

• The photo and audio words were found using the "sensor_fusion_segmentor()" routine.

• The photo and audio words were then stripped out and processed using the "process_photo()" and process_mic()" routines.

• The photo and audio words were then normalized with respect to their energy content using the "normalize()" routine.

• The photo and audio templates with the overall lowest DTW scores were compared using the decision theory previously described above for sensor fusion.

Appendix C. Computer Software

"Speech.c" Source Program

```
/*-----------------------------------------------------------------

ASR test program demonstrating the usefulness of using an IR and/or visible sensor
for speech recognition. Mostly written in portable C code.

Program: speech.c
Programmer: Patrick T. Marshall
Date: 02/25/90
Organization: WRDC/AAWP-2,
        WPAFB, OH 45433
Phone: (513) 255-2471

-------------------------------------------------------------------*/
/*****************************************************************/
/*                                                               */
/*      Note: The constant varibles "MAXDRIVE, MAXEXT,           */
/*      ect." are located in the "dir h" include file            */
/*      which is included in "speech.h".                         */
/*                                                               */
/***********************************************************/
#include "c:\borlandc\thesis\ peech.h"

/* Function prototypes */
int alloc_dma_buf(void);        /* Allocate dma buffers */
void intr_setup(void);          /* Set up interrupt operation */
void dma_setup(void);                   /* Set up dma operation */
void dma_finish(void);          /* Called via atexit() mechanism */
void interrupt far dma_isr(void);
void start_dma(char far *,unsigned); /* Start a dma operation */
void init_brd(void);            /* Initialize A/D board */
void on_brd(void);              /* Turn A/D board on */
void set_up_plt(char[25],double,double,double,double,char[3]);
void plot(float far *,float,float,unsigned long,int);
void label_plot(label_struct[16],int);
void print_plot(void);
void start_plot(void);
void erase_plot(void);
void init_plot(void);
void data_acquisition(void);
void acquire_data(void);
void create_temps(void);
void speech_rec(void);
void start_processing_data(void);
void fft2(float far *,float far *,unsigned,int);
void plot_data(void);
void plot_info(void);
void plot_time(char,unsigned);
void plot_freq(char);
/*
get_file_info(char[25],unsigned*,unsigned long *);
convert_data(float *,unsigned long,unsigned long,char[25]);
*/
void average_template(void);

/*                                                              */
/*       Begin main function                                    */
/*                                                              */
main()
{
  int display_setup_screen(void);
  void data_acquisition();
  void plot_data();
  int read_string (int, int, int, int, char*, char*, int);
  char ask_question (int, int. char*, char*);
  void wait_message (int, int, char*   har*);
  void process_acq_menu (int, int);
  void message (int, int, char*, char*);
  void clear_message (void);
  logical file_exists (char*);
  void build_path (char*, char, char*, int);

  unsigned char key;

  clrscr();
  cursor_off();
  display_setup_screen();
  e_xit = FALSE;
  do
  {
       key = read_key();
       switch (key)
       {
         case F1:
              data_acquisition(); /* Controls DMA data transfer */
              e_xit = FALSE;
              break;
         case F2:
                plot_data();
                cursor_off();
                e_xit = FALSE;
                break;
            case F3:
                create_temps();
                e_xit = FALSE;
                break;
            case F4:
                speech_rec();
                e_xit = FALSE;
                break;
            case ESC:
                e_xit = TRUE;
                break;
        }
    } while (! e_xit);
    cursor_on();
}                           /* End of main */


display_setup_screen()
{
 char disk_space_buf[33],memory[33];
 struct dfree disk;
 unsigned long disk_space,max_par;

 clrscr();
 normal_video();
 gotoxy (28,1);
 cputs  ("SPEECH RECOGNITION PROGRAM");
 create_window(DOUBLE,25,3,55,16);
 gotoxy (6,1);
 textcolor (LIGHTRED);
 cputs ("F1");
 textcolor (LIGHTGRAY);
 cputs ("-Data Acquisition  ");
 gotoxy (6,3);
 textcolor (LIGHTRED);
 cputs ("F2");
 textcolor (LIGHTGRAY);
 cputs ("-Plot Data      ");
 gotoxy (6,5);
 textcolor (LIGHTRED);
 cputs ("F3");
 textcolor (LIGHTGRAY);
 cputs ("-Create Templates   ");
 gotoxy (6,7);
 textcolor (LIGHTRED);
 cputs ("F4");
 textcolor (LIGHTGRAY);
 cputs ("-Speech Recognition" );
 gotoxy (2,9);
 getdfree(0,&disk);
 disk_space = (long)disk.df_avail*(long)disk.df_sclus*
 (long)disk.df_bsec;
 ultoa(disk_space,disk_space_buf,10);
 cputs ("Disk space: ");
 textcolor (LIGHTGRAY);
 cputs (disk_space_buf);
 cputs (" bytes");
 gotoxy (2,11);
 max_par=allocmem(0xffff, &seg);
 tol_mem_avail = max_par << 4;
 ultoa(tol_mem_avail,memory,10);
 cputs ("Memory: ");
 textcolor (LIGHTGRAY);
 cputs (memory);
 cputs (" bytes");
 return;
}

void data_acquisition()
{
 void process_acq_menu (int, int);
 unsigned bar, active_bar=1;
 unsigned char key;

 create_window(DOUBLE,17,12,63,21);
 gotoxy (1,1);
 cputs("Start acquisition?");
 gotoxy (1,2);
 cputs("File Name: ");
 gotoxy (1,3);
 cputs("Window size: (secs)");
 gotoxy (1,4);
 cputs  ("Number of different runs:");
 gotoxy (1,5);
 cputs  ("Number of similar runs:");
 gotoxy (1,6);
```

```
cputs ("Isolated or continuous:");
gotoxy (1,7);
cputs ("Default Drive:");
gotoxy (1,8);
cputs ("Path Name:");
for (bar=1; bar<=8; bar++)
  process_acq_menu (bar, NORMAL);
bar = 1;
process_acq_menu (active_bar, HIGHLIGHT);
cursor_off();
e_xit = FALSE;
do
{
  key = read_key();
  switch (key)
  {
    case UP_ARROW:
         if (bar > 1)
         {
          process_acq_menu (bar, NORMAL);
          bar--;
          process_acq_menu (bar, HIGHLIGHT);
         }
         break;
    case DOWN_ARROW:
         if (bar < 8)
         {
          process_acq_menu (bar, NORMAL);
          bar++;
          process_acq_menu (bar, HIGHLIGHT);
         }
         break;
    case CR:
         process_acq_menu (bar, ACTIVE);
         break;
    case ESC:
         e_xit = TRUE;
         break;
   }
 } while (! e_xit);
if (graphics == FALSE) restore_window();
graphics = FALSE;
return;
}
```

```
void process_acq_menu (int bar, int mode)
{
 void read_float (float*, float, float, int);
 void read_long  (long*, long, long, int);
 void read_int   (int*, int, int, int);
 void read_char  (char*, char, char);
 int  read_string (int, int, int, int, char*, char*, int);
 char ask_question (int, int, char*, char*);
 logical file_exists (char*);
 unsigned long count;

 count = (unsigned long) 16*allocmem(0xffff,&segadd);
 max_ts = (float)count/(2.0*(25000.0+2500.0)); /* In secs */
 min_ts = 0.0;
 if (init_flag)
 {
  ts = max_ts;
  status = 1;
  init_flag = OFF;
 }
 if (mode == ACTIVE) switch (bar)
 {
  case 1: restore_window();
          restore_window();
          strcpy(path,dir);
          strcat(path,dat_file_name);
          if((op_mode[0] == 'C') && (num_words > 5)) num_words = 5;
          if((op_mode[0] == 'I') && (num_words > 12)) num_words = 12;
          acquire_data();
          e_xit = TRUE;
          graphics = TRUE;
          display_setup_screen();
          return;
  case 2: read_string (1,15,80,18,"New file name:",dat_file_name,MAXFILE-1);
          restore_window();
          gotoxy (30,2);
          clreol();
          break;
  case 3: read_float (&ts,min_ts,max_ts,10);
          gotoxy (30,3);
          clreol();
          break;
  case 4: read_int(&num_words,min_words,max_words,word_size);
          gotoxy (30,4);
```

```
                clreol();
                break;
    case 5:   read_int (&num_runs,min_runs,max_runs,run_size);
                gotoxy (30,5);
                clreol();
                break;
    case 6:   if(op_mode[0] == 'I') op_mode[0] = 'C';
                else op_mode[0] = 'I';
                gotoxy (30,6);
                clreol();
                break;
    case 7:   read_string (1,15,80,18,"New drive:",drive,MAXDRIVE-1);
                restore_window();
                gotoxy (30,7);
                clreol();
                break;
    case 8:   read_string (1,15,80,18,"New path name:",dir,MAXDIR-1);
                restore_window();
                gotoxy (28,8);
                clreol();
                break;
}
if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
switch (bar)
{
  case 1: gotoxy (30,1);
             cprintf("%s", "YES");
             break;
  case 2: gotoxy (30,2);
             cprintf("%s", dat_file_name);
             break;
  case 3: gotoxy (30,3);
             cprintf("%f", ts);
             break;
  case 4: gotoxy (34,4);
             cprintf("%3.3d", num_words);
             break;
  case 5: gotoxy (34,5);
             cprintf("%3.3d", num_runs);
             break;
  case 6: gotoxy (30,6);
             if(op_mode[0] == 'C') cprintf("%s", "Continuous");
             else cprintf("%s", "Isolated");
             break;
  case 7: gotoxy (35,7);
```

```
          cprintf("%s:", drive);
          break;
   case 8: gotoxy (28,8);
          cprintf("%s", dir);
          break;
 }
 normal_video();
 return;
}


void plot_data()
{
 void process_plot_menu (int, int);
 unsigned bar,active_bar=1;
 unsigned char key;

 max_ts = 0.8;
 create_window(DOUBLE,15,6,65,13);
 gotoxy (1,1);
 cputs("Start plotting:");
 gotoxy (1,2);
 cputs("Plot information:");
 gotoxy (1,3);
 cputs("Plot photo time data:");
 gotoxy (1,4);
 cputs("Plot mic time data:");
 gotoxy (1,5);
 cputs("Plot photo freq data:");
 gotoxy (1,6);
 cputs("Plot mic freq data:");
 for (bar=1; bar<=6; bar++)
   process_plot_menu (bar, NORMAL);
 bar = 1;
 process_plot_menu (active_bar, HIGHLIGHT);
 cursor_off();
 e_xit = FALSE;
 do
 {
  key = read_key();
  switch (key)
  {
   case UP_ARROW:
        if (bar > 1)
        {
         process_plot_menu (bar, NORMAL);
         bar--;
         process_plot_menu (bar, HIGHLIGHT);
        }
        break;
    case DOWN_ARROW:
        if (bar < 6)
        {
         process_plot_menu (bar, NORMAL);
         bar++;
         process_plot_menu (bar, HIGHLIGHT);
        }
        break;
    case CR:
        process_plot_menu (bar, ACTIVE);
        break;
    case ESC:
        e_xit = TRUE;
        break;
    }
  } while (! e_xit);
  if (graphics == FALSE) restore_window();
  graphics = FALSE;
  return;
}


void process_plot_menu(int bar, int mode)
{
  void read_char  (char*, char, char);
  void photo_freq_info(void);
  void mic_freq_info(void);
  int  read_string (int, int, int, int, char*, char*, int);
  logical file_exists (char*);

  e_xit = FALSE:
  if (mode == ACTIVE) switch (bar)
  {
   case 1: if ((*plot_photo_time_flag == 'y') ||
                 (*plot_photo_time_flag == 'Y') ||
                 (*plot_mic_time_flag == 'y')   ||
                 (*plot_mic_time_flag == 'Y')   ||
                 (*plot_photo_freq_flag == 'y') ||
                 (*plot_photo_freq_flag == 'Y') ||
            (*plot_mic_freq_flag == 'y')  ||
            (*plot_mic_freq_flag == 'Y'))
        {
         restore_window();
         restore_window();
         strcpy(path,dir);
         strcat(path,dat_file_name);
         start_plot();
         e_xit = TRUE;
         graphics = TRUE;
         display_setup_screen();
        }
        break;
case 2: plot_info();
        restore_window();
        gotoxy (30,2);
        clreol();
        break;
case 3: if ((*plot_photo_time_flag == 'y') || (*plot_photo_time_flag == 'Y'))
         strcpy(&plot_photo_time_flag[0],"NO");
        else
         strcpy(&plot_photo_time_flag[0],"YES");
        gotoxy (30,3);
        clreol();
        break;
case 4: if ((*plot_mic_time_flag == 'y') || (*plot_mic_time_flag == 'Y'))
         strcpy(&plot_mic_time_flag[0],"NO");
        else
         strcpy(&plot_mic_time_flag[0],"YES");
        gotoxy (30,4);
        clreol();
        break;
case 5: if ((*plot_photo_freq_flag== 'y') || (*plot_photo_freq_flag== 'Y'))
         strcpy(&plot_photo_freq_flag[0],"NO");
        else
        {
         photo_freq_info();
         restore_window();
         strcpy(&plot_photo_freq_flag[0],"YES");
        }
        gotoxy (30,5);
        clreol();
        break;
case 6: if ((*plot_mic_freq_flag== 'y') || (*plot_mic_freq_flag== 'Y'))
         strcpy(&plot_mic_freq_flag[0],"NO");
```

```
          else
          {
           mic_freq_info();
           restore_window();
           strcpy(&plot_mic_freq_flag[0],"YES");
          }
          gotoxy (30,6);
          clreol();
          break;
  }
  if (e_xit == FALSE)
  {
   if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
   switch (bar)
   {
    case 1: gotoxy (35,1);
            cprintf("%s", "YES");
            break;
    case 2: gotoxy (35,2);
            if(def_flag == TRUE)
               cprintf("%s", "DEFAULT");
            else
               cprintf("%s", "CHANGED");
            break;
    case 3: gotoxy (35,3);
            cprintf("%s", plot_photo_time_flag);
            break;
    case 4: gotoxy (35,4);
            cprintf("%s", plot_mic_time_flag);
            break;
    case 5: gotoxy (35,5);
            cprintf("%s", plot_photo_freq_flag);
            break;
    case 6: gotoxy (35,6);
            cprintf("%s", plot_mic_freq_flag);
            break;
   }
   normal_video();
  }
 return;
}


void plot_info()
{
```

```
void process_pi_menu (int, int);
unsigned bar,active_bar=1;
unsigned char key;

max_ts = 0.8:
create_window(DOUBLE,15,12,65,24);
gotoxy (1,1);
cputs("Start time (sec):");
gotoxy (1,2);
cputs("Stop time (sec):");
gotoxy (1,3);
cputs("Beginning word number:");
gotoxy (1,4);
cputs("Ending word number:");
gotoxy (1,5);
cputs("Beginning word multiple:");
gotoxy (1,6);
cputs("Ending word multiple:");
gotoxy (1,7);
cputs("Plot templates:");
gotoxy (1,8);
cputs("Plot mic energy:");
gotoxy (1,9);
cputs("Data file name:");
gotoxy (1,10);
cputs("Default Drive:");
gotoxy (1,11);
cputs  ("Path Name:");
for (bar=1; bar<=11; bar++)
  process_pi_menu (bar, NORMAL);
bar = 1;
process_pi_menu (active_bar, HIGHLIGHT);
cursor_off();
e_xit = FALSE;
do
{
  key = read_key();
  switch (key)
  {
    case UP_ARROW:
         if (bar > 1)
         {
          process_pi_menu (bar, NORMAL);
          bar--;
          process_pi_menu (bar, HIGHLIGHT);
          }
          break;
      case DOWN_ARROW:
          if (bar < 11)
          {
            process_pi_menu (bar, NORMAL);
            bar++;
            process_pi_menu (bar, HIGHLIGHT);
          }
          break;
      case CR:
          process_pi_menu (bar, ACTIVE);
          break;
      case ESC:
          e_xit = TRUE;
          break;
    }
  } while (! e_xit);
  e_xit = FALSE;
  graphics = FALSE;
  return;
}


void process_pi_menu(int bar, int mode)
{
  void read_float (float*, float, float, int);
  void read_long  (long*, long, long, int);
  void read_int   (int*, int, int, int);
  void read_char  (char*, char, char);
  int  read_string (int, int, int, int, char*, char*, int);
  logical file_exists (char*);
  float prev_val1;long prev_val2;int prev_val3;char prev_val4[MAXFILE];

  e_xit = FALSE;
  if (mode == ACTIVE) switch (bar)
  {
    case 1: prev_val1 = start_time;
            read_float(&start_time,min_time,max_time,4);
            if(prev_val1 != start_time) def_flag = FALSE;
            gotoxy (30,1);
            clreol();
            break;
    case 2: prev_val1 = stop_time;
            read_float(&stop_time,min_time,max_time,4);
          if(prev_val1 != start_time) def_flag = FALSE;
          gotoxy (30,2);
          clreol();
          break;
case 3: prev_val3 = beg_word;
          read_int (&beg_word,0,max_words,4);
          gotoxy (30,3);
          clreol();
          break;
case 4: prev_val3 = beg_word;
          read_int (&end_word,0,max_words,4);
          if(prev_val3 != end_word) def_flag = FALSE;
          gotoxy (30,4);
          clreol();
          break;
case 5: prev_val3 = beg_mult;
          read_int (&beg_mult,0,max_plots,4);
          if(prev_val3 != beg_mult) def_flag = FALSE;
          gotoxy (30,5);
          clreol();
          break;
case 6: prev_val3 = end_mult;
          read_int (&end_mult,0,max_plots,4);
          if(prev_val3 != end_mult) def_flag = FALSE;
          gotoxy (30,6);
          clreol();
          break;
case 7: strcpy(prev_val4,plot_tmps);
          if ((*plot_tmps== 'y') || (*plot_tmps== 'Y'))
            strcpy(&plot_tmps[0],"NO");
          else
          {
            strcpy(&plot_tmps[0],"YES");
            beg_mult = 1;end_mult = 1;
          }
          if(*prev_val4 != *plot_tmps) def_flag = FALSE;
          strcpy(dat_path,dir);
          strcat(dat_path,dat_file_name);
          gotoxy (30,7);
          clreol();
          break;
case 8: if(plot_mic_energy_flag[0] == 'Y') strcpy(&plot_mic_energy_flag[0],"NO");
          else strcpy(&plot_mic_energy_flag[0],"YES");
          gotoxy (30,8);
          clreol();
```

```
            break;
    case 9: strcpy(prev_val4,dat_file_name);
            read_string (1,15,80,18,"New file name:",dat_file_name,MAXFILE-1);
            if(prev_val4 != dat_file_name) def_flag = FALSE;
            restore_window();
            gotoxy (30,9);
            clreol();
            break;
    case 10:strcpy(prev_val4,drive);
            read_string (1,15,80,18,"New drive:",drive,MAXDRIVE-1);
            if(prev_val4 != drive) def_flag = FALSE;
            restore_window();
            gotoxy (30,10);
            clreol();
            break;
    case 11:strcpy(prev_val4,dir);
            read_string (1,15,80,18,"New path name:",dir,MAXDIR-1);
            restore_window();
            if(prev_val4 != dir) def_flag = FALSE;
            gotoxy (30,11);
            clreol();
            break;
  }
  if (e_xit == FALSE)
  {
   if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
   switch (bar)
   {
    case 1: gotoxy (35,1);
            cprintf("%f", start_time);
            break;
    case 2: gotoxy (35,2);
            cprintf("%f", stop_time);
            break;
    case 3: gotoxy (35,3);
            cprintf("%d", beg_word);
            break;
    case 4: gotoxy (35,4);
            cprintf("%d", end_word);
            break;
    case 5: gotoxy (35,5);
            cprintf("%d", beg_mult);
            break;
    case 6: gotoxy (35,6);
            cprintf("%d", end_mult);
            break;
   case 7: gotoxy (35,7);
            cprintf("%s", plot_tmps);
            break;
   case 8: gotoxy (35,8);
            cprintf("%s", plot_mic_energy_flag);
            break;
   case 9: gotoxy (32,9);
            cprintf("%s", dat_file_name);
            break;
   case 10:gotoxy (35,10);
            cprintf("%s:", drive);
            break;
   case 11:gotoxy (30,11);
            cprintf("%s", dir);
            break;
  }
  normal_video();
 }
 return;
}


void photo_freq_info()
{
 void process_photo_freq_info_menu(int, int);
 unsigned bar,active_bar=1;
 unsigned char key;

 max_ts = 0.8;
 create_window(DOUBLE,15,10,65,14);
 gotoxy (1,1);
 cputs("Start Photo freq (Hz)?");
 gotoxy (1,2);
 cputs("Ending Photo freq (Hz)?");
 gotoxy (1,3);
 cputs("Max Photo freq amplitude?");
 for (bar=1; bar<=3; bar++)
   process_photo_freq_info_menu(bar, NORMAL);
 bar = 1;
 process_photo_freq_info_menu(active_bar, HIGHLIGHT);
 cursor_off();
 e_xit = FALSE;
 do
 {
  key = read_key();
  switch (key)
  {
   case UP_ARROW:
        if (bar > 1)
        {
          process_photo_freq_info_menu(bar, NORMAL);
          bar--;
          process_photo_freq_info_menu(bar, HIGHLIGHT);
        }
        break;
   case DOWN_ARROW:
        if (bar < 6)
        {
          process_photo_freq_info_menu(bar, NORMAL);
          bar++;
          process_photo_freq_info_menu(bar, HIGHLIGHT);
        }
        break;
   case CR:
        process_photo_freq_info_menu(bar, ACTIVE);
        break;
   case ESC:
        e_xit = TRUE;
        break;
  }
 } while (! e_xit);
 e_xit = FALSE;
 graphics = FALSE;
 return;
}

void process_photo_freq_info_menu(int bar, int mode)
{
 void read_float (float*, float, float, int);
 void read_long  (long*, long, long, int);
 void read_int   (int*, int, int, int);
 void read_char  (char*, char, char);
 int  read_string (int, int, int, int, char*, char*, int);
 logical file_exists (char*);

 e_xit = FALSE;
 if (mode == ACTIVE) switch (bar)
 {
  case 1: read_float(&start_photo_freq,min_freq,max_freq,4);
```

```
            gotoxy (30,1);
            clreol();
            break;
   case 2: read_float(&end_photo_freq,min_freq,max_freq,4);
            gotoxy (30,2);
            clreol();
            break;
   case 3: read_float(&max_photo_amp,min_amp,max_amp,4);
            gotoxy (30,3);
            clreol();
            break;
  }
 if (e_xit == FALSE)
 {
  if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
  switch (bar)
  {
   case 1: gotoxy (35,1);
            cprintf("%f", start_photo_freq);
            break;
   case 2: gotoxy (35,2);
            cprintf("%f", end_photo_freq);
            break;
   case 3: gotoxy (35,3);
            cprintf("%f", max_photo_amp);
            break;
  }
  normal_video();
 }
 return;
}


void mic_freq_info()
{
 void process_mic_freq_info_menu(int, int);
 unsigned bar,active_bar=1;
 unsigned char key;

 max_ts = 0.8;
 create_window(DOUBLE,15,10,65,14);
 gotoxy (1,1);
 cputs("Start MIC freq (Hz)?");
 gotoxy (1,2);
 cputs("Ending MIC freq (Hz)?");
```

```
gotoxy (1,3);
cputs("Max MIC freq amplitude?");
for (bar=1; bar<=3; bar++)
  process_mic_freq_info_menu(bar, NORMAL);
bar = 1;
process_mic_freq_info_menu(active_bar, HIGHLIGHT);
cursor_off();
e_xit = FALSE;
do
{
  key = read_key();
  switch (key)
  {
   case UP_ARROW:
        if (bar > 1)
        {
          process_mic_freq_info_menu(bar, NORMAL);
          bar--;
          process_mic_freq_info_menu(bar, HIGHLIGHT);
        }
        break;
   case DOWN_ARROW:
        if (bar < 3)
        {
          process_mic_freq_info_menu(bar, NORMAL);
          bar++;
          process_mic_freq_info_menu(bar, HIGHLIGHT);
        }
        break;
   case CR:
        process_mic_freq_info_menu(bar, ACTIVE);
        break;
   case ESC:
        e_xit = TRUE;
        break;
  }
 } while (! e_xit);
 e_xit = FALSE;
 graphics = FALSE;
 return;
}

void process_mic_freq_info_menu(int bar, int mode)
{
  void read_float (float*, float, float, int);
void read_long  (long*, long, long, int);
void read_int   (int*, int, int, int);
void read_char  (char*, char, char);
int  read_string (int, int, int, int, char*, char*, int);
logical file_exists (char*);

e_xit = FALSE;
if (mode == ACTIVE) switch (bar)
{
 case 1: read_float(&start_mic_freq,min_freq,max_freq,4);
         gotoxy (30,1);
         clreol();
         break;
 case 2: read_float(&end_mic_freq,min_freq,max_freq,4);
         gotcxy (30,2);
         clreol();
         break;
 case 3: read_float(&max_mic_amp,min_amp,max_amp,4);
         gotoxy (30,3);
         clreol();
         break;
}
if (e_xit == FALSE)
{
 if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
 switch (bar)
 {
  case 1: gotoxy (35,1);
          cprintf("%f", start_mic_freq);
          break;
  case 2: gotoxy (35,2);
          cprintf("%f", end_mic_freq);
          break;
  case 3: gotoxy (35,3);
          cprintf("%f", max_mic_amp);
          break;
 }
 normal_video();
}
return;
}


void create_temps()
{
```

```
void process_template_info_menu (int, int);
unsigned bar,active_bar=1;
unsigned char key;

max_ts = 0.8;
create_window(DOUBLE,15,9,65,24);
gotoxy (1,1);
cputs("Start template processing:");
gotoxy (1,2);
cputs("Start time (sec):");
gotoxy (1,3);
cputs("Stop time (sec):");
gotoxy (1,4);
cputs("Beginning word number:");
gotoxy (1,5);
cputs("Ending word number:");
gotoxy (1,6);
cputs("Beginning word multiple:");
gotoxy (1,7);
cputs("Ending word multiple:");
gotoxy (1,8);
cputs("DTW window size:");
gotoxy (1,9);
cputs("Max slope step size:");
gotoxy (1,10);
cputs("Average templates?");
gotoxy (1,11);
cputs("Data file name:");
gotoxy (1,12);
cputs("Template file name:");
gotoxy (1,13);
cputs("Default Drive:");
gotoxy (1,14);
cputs  ("Path Name:");
for (bar=1; bar<=14; bar++)
  process_template_info_menu (bar, NORMAL);
bar = 1;
process_template_info_menu (active_bar, HIGHLIGHT);
cursor_off();
e_xit = FALSE;
do
{
  key = read_key();
  switch (key)
  {
     case UP_ARROW:
         if (bar > 1)
         {
          process_template_info_menu (bar, NORMAL);
          bar--;
          process_template_info_menu (bar, HIGHLIGHT);
         }
         break;
     case DOWN_ARROW:
         if (bar < 14)
         {
          process_template_info_menu (bar, NORMAL);
          bar++;
          process_template_info_menu (bar, HIGHLIGHT);
         }
         break;
     case CR:
         process_template_info_menu (bar, ACTIVE);
         break;
     case ESC:
         e_xit = TRUE;
         break;
     }
  } while (! e_xit);
  e_xit = FALSE;
  graphics = FALSE;
  restore_window();
  return;
}


void process_template_info_menu(int bar, int mode)
{
 void read_float (float*, float, float, int);
 void read_long  (long*, long, long, int);
 void read_int   (int*, int, int, int);
 void read_char  (char*, char, char);
 int  read_string (int, int, int, int, char*, char*, int);
 void average_templates(void);
 void create_templates(void);
 logical file_exists (char*);

 e_xit = FALSE;
 if (mode == ACTIVE) switch (bar)
 {
    case 1: restore_window();restore_window();
            strcpy(dat_path,dir);strcat(dat_path,dat_file_name);
            strcpy(tmp_path,dir);strcat(tmp_path,tmp_file_name);
            if(end_word>11) end_word = 11;
            clrscr();
            printf("        Processing templates ... please wait!\n");
            printf("\n");
            if(best_template_flag[0] == 'N') create_templates();
            else find_best_templates();

stop_time = 1.0;
strcpy(dat_patn,dir);strcat(dat_path,"janet");
strcpy(tmp_path,dir);strcat(tmp_path,"janet");
create_templates();
stop_time = 1.0;
strcpy(dat_path,dir);strcat(dat_path,"don");
strcpy(tmp_path,dir);strcat(tmp_path,"don");
create_templates();
stop_time = 1.5;
strcpy(dat_path,dir);strcat(dat_path,"jill");
strcpy(tmp_path,dir);strcat(tmp_path,"jill");
create_templates();
stop_time = 1.0;
strcpy(dat_path,dir);strcat(dat_path,"mary");
strcpy(tmp_path,dir);strcat(tmp_path,"mary");
create_templates();

            e_xit = TRUE;graphics = TRUE;display_setup_screen();
            normal_video();break;
    case 2: read_float(&start_time,min_time,max_time,4);
            gotoxy (30,bar);clreol();break;
    case 3: read_float(&stop_time,min_time,max_time,4);
            gotoxy (30,bar);clreol();break;
    case 4: read_int (&beg_word,0,max_words,4);
            gotoxy (30,bar);clreol();break;
    case 5: read_int (&end_word,0,max_words,4);
            gotoxy (30,bar);clreol();break;
    case 6: read_int (&beg_mult,0,max_plots,4);
            gotoxy (30,bar);clreol();break;
    case 7: read_int (&end_mult,0,max_plots,4);
            gotoxy (30,bar);clreol();break;
    case 8: read_int (&window_width,1,20,4);
            gotoxy (30,bar);clreol();break;
    case 9: read_int (&max_step,1,20,4);
            gotoxy (30,bar);clreol();break;
```

```
  case 10:if ((*best_template_flag == 'y') || (*best_template_flag == 'Y'))
            strcpy(&best_template_flag[0],"NO");
          else
            strcpy(&best_template_flag[0],"YES");
          gotoxy (30,bar);clreol();break;
  case 11:read_string (1,15,80,18,"New file name:",dat_file_name,MAXFILE-1);
          restore_window();gotoxy (30,bar);clreol();break;
  case 12:read_string (1,15,80,18,"New file name:",tmp_file_name,MAXFILE-1);
          restore_window();gotoxy (30,bar);clreol();break;
  case 13:read_string (1,15,80,18,"New drive:",drive,MAXDRIVE-1);
          restore_window();gotoxy (30,bar);clreol();break;
  case 14:read_string (1,15,80,18,"New path name:",dir,MAXDIR-1);
          restore_window();gotoxy (30,bar);clreol();break;
 }
 if (e_xit == FALSE)
 {
  if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
  switch (bar)
  {
   case 1: gotoxy (35,bar);cprintf("%s","YES");break;
   case 2: gotoxy (35,bar);cprintf("%f", start_time);break;
   case 3: gotoxy (35,bar);cprintf("%f", stop_time);break;
   case 4: gotoxy (35,bar);cprintf("%d", beg_word);break;
   case 5: gotoxy (35,bar);cprintf("%d", end_word);break;
   case 6: gotoxy (35,bar);cprintf("%d", beg_mult);break;
   case 7: gotoxy (35,bar);cprintf("%d", end_mult);break;
   case 8: gotoxy (35,bar);cprintf("%d", window_width);break;
   case 9: gotoxy (35,bar);cprintf("%d", max_step);break;
   case 10:gotoxy (35,bar);cprintf("%s",best_template_flag);break;
   case 11:gotoxy (32,bar);cprintf("%s", dat_file_name);break;
   case 12:gotoxy (32,bar);cprintf("%s", tmp_file_name);break;
   case 13:gotoxy (35,bar);cprintf("%s:", drive);break;
   case 14:gotoxy (30,bar);cprintf("%s", dir);break;
  }
  normal_video();
 }
 return;
}


void speech_rec()
{
 void process_spch_rec_info(int, int);
 unsigned bar,active_bar=1;
unsigned cha

max_ts = 0.8;max_step = 4;window_width = 3; /* Default constants */
create_window(DOUBLE,15,10,65,25);
gotoxy (1,1);
cputs("Start recognition process:");
gotoxy (1,2);
cputs("Beginning word number:");
gotoxy (1,3);
cputs("Ending word number:");
gotoxy (1,4);
cputs("Beginning word multiple:");
gotoxy (1,5);
cputs("Ending word multiple:");
gotoxy (1,6);
cputs("Beginning template number:");
gotoxy (1,7);
cputs("Ending template number:");
gotoxy (1,8);
cputs("DTW window size:");
gotoxy (1,9);
cputs("Max slope step size:");
gotoxy (1,10);
cputs("Data file name:");
gotoxy (1,11);
cputs("Template file name:");
gotoxy (1,12);
cputs("Default Drive:");
gotoxy (1,13);
cputs  ("Path Name:");
gotoxy (1,14);
cputs  ("Sensor fusion:");
for (bar=1; bar<=14; bar++)
  process_spch_rec_info(bar, NORMAL);
bar = 1;
process_spch_rec_info (active_bar, HIGHLIGHT);
cursor_off();
e_xit = FALSE;
do
{
  key = read_key();
  switch (key)
  {
    case UP_ARROW:
        if (bar > 1)
```

```
          {
           process_spch_rec_info (bar, NORMAL);
           bar--;
           process_spch_rec_info (bar, HIGHLIGHT);
          }
          break;
      case DOWN_ARROW:
          if (bar < 14)
          {
           process_spch_rec_info (bar, NORMAL);
           bar++;
           process_spch_rec_info (bar, HIGHLIGHT);
          }
          break;
      case CR:
          process_spch_rec_info (bar, ACTIVE);
          break;
      case ESC:
          e_xit = TRUE;
          restore_window();
          break;
     }
   } while (! e_xit);
   normal_video();
   e_xit = FALSE;
   graphics = FALSE;
   return;
  }


void process_spch_rec_info(int bar, int mode)
{
  void read_float (float*, float, float, int);
  void read_long  (long*, long, long, int);
  void read_int   (int*, int, int, int);
  void read_char  (char*, char, char);
  int  read_string (int, int, int, int, char*, char*, int);
  logical file_exists(char*);
  e_xit = FALSE;
  if (mode == ACTIVE) switch (bar)
  {
   case 1: restore_window();
           restore_window();
           strcpy(dat_path,dir);
           strcat(dat_path,dat_file_name);
```

```
            strcpy(tmp_path,dir);
            strcat(tmp_path,tmp_file_name);
            clrscr();
            system("del c:\\borlandc\\thesis\\results.dat");
            start_speech_rec(sensor_fusion_flag);

strcpy(dat_path,dir);strcat(dat_path,"janet");
strcpy(tmp_path,dir);strcat(tmp_path,"janet");
start_speech_rec(sensor_fusion_flag);
strcpy(dat_path,dir);strcat(dat_path,"don");
strcpy(tmp_path,dir);strcat(tmp_path,"don");
start_speech_rec(sensor_fusion_flag);
strcpy(dat_path,dir);strcat(dat_path,"mary");
strcpy(tmp_path,dir);strcat(tmp_path,"mary");
start_speech_rec(sensor_fusion_flag);
strcpy(dat_path,dir);strcat(dat_path,"jill");
strcpy(tmp_path,dir);strcat(tmp_path,"jill");
start_speech_rec(sensor_fusion_flag);

            e_xit = TRUE;
            graphics = TRUE;
            break;
   case 2: read_int (&beg_word,0,17,4);
            gotoxy (30,2);
            clreol();
            break;
   case 3: read_int (&end_word,0,17,4);
            gotoxy (30,3);
            clreol();
            break;
   case 4: read_int (&beg_mult,0,max_plots,4);
            gotoxy (30,4);
            clreol();
            break;
   case 5: read_int (&end_mult,0,max_plots,4);
            gotoxy (30,5);
            clreol();
            break;
   case 6: read_int (&beg_template_num,0,11,4);
            gotoxy (30,6);
            clreol();
            break;
   case 7: read_int (&end_template_num,0,11,4);
            gotoxy (30,7);
            clreol();
```

```
          break;
case 8: read_int (&window_width,1,20,4);
          gotoxy (30,8);
          clreol();
          break;
case 9: read_int (&max_step,1,20,4);
          gotoxy (30,9);
          clreol();
          break;
case 10:read_string (1,15,80,18,"Data file name:",dat_file_name,MAXFILE-1);
          restore_window();
          gotoxy (30,10);
          clreol();
          break;
case 11:read_string (1,15,80,18,"Template file name:",tmp_file_name,MAXFILE-1);
          restore_window();
          gotoxy (30,11);
          clreol();
          break;
case 12:read_string (1,15,80,18,"New drive:",drive,MAXDRIVE-1);
          restore_window();
          gotoxy (30,12);
          clreol();
          break;
case 13:read_string (1,15,80,18,"New path name:",dir,MAXDIR-1);
          restore_window();
          gotoxy (30,13);
          clreol();
          break;
case 14:if(sensor_fusion_flag == FALSE) sensor_fusion_flag = TRUE;
          else sensor_fusion_flag = FALSE;
          gotoxy (30,14);
          clreol();
          break;
}
if (e_xit == FALSE)
{
 if (mode == HIGHLIGHT || mode == ACTIVE) inverse_video();
 switch (bar)
 {
  case 1: gotoxy (35,1);
            cprintf("%s","YES");
            break;
  case 2: gotoxy (35,2);
            cprintf("%d", beg_word);
            break;
      case 3: gotoxy (35,3);
            cprintf("%d", end_word);
            break;
      case 4: gotoxy (35,4);
            cprintf("%d", beg_mult);
            break;
      case 5: gotoxy (35,5);
            cprintf("%d", end_mult);
            break;
      case 6: gotoxy (35,6);
            cprintf("%d", beg_template_num);
            break;
      case 7: gotoxy (35,7);
            cprintf("%d", end_template_num);
            break;
      case 8: gotoxy (35,8);
            cprintf("%d", window_width);
            break;
      case 9: gotoxy (35,9);
            cprintf("%d", max_step);
            break;
      case 10:gotoxy (35,10);
            cprintf("%s", dat_file_name);
            break;
      case 11:gotoxy (35,11);
            cprintf("%s", tmp_file_name);
            break;
      case 12:gotoxy (35,12);
            cprintf("%s:", drive);
            break;
      case 13:gotoxy (30,13);
            cprintf("%s", dir);
            break;
      case 14:gotoxy (35,14);
            if(sensor_fusion_flag == FALSE) cprintf("%s","NO");
            else cprintf("%s","YES");
            break;
     }
     normal_video();
   }
  return;
}
```

"Plt_spc.c" Source Program

```
/*---------------------------------------------------------------

Program contains miscellaneous plot functions used by main.

Program: plt_spc.c
Programmer: Patrick T. Marshall
Date: 2/25/91
Organization: WRDC/AAWP-2,
              WPAFB, OH 45433
Phone: (513) 255-2471

---------------------------------------------------------------*/
#pragma check_stack(off)
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\plot.h"
#include "c:\borlandc\thesis\misc_spch.h"

/* Function prototypes */
void set_up_plt(char[25],double,double,double,double,char[3]);
void plot(float huge *,float,float,unsigned long,int);
void label_plot(label_struct[],int,char[]);
void print_plot(void);
void start_plot(void);
void erase_plot(void);
void init_plot(void);
void fft2(float huge *,float huge *,unsigned,int);
void plot_data(void);
void plot_time(char,unsigned);
void plot_freq(char);
void plot_photo_tmps(char,unsigned);
void plot_mic_energy(unsigned);
float huge *convert_data(unsigned *,unsigned,char[],unsigned long *);
get_file_info(unsigned*,unsigned long *);
char ask_question (int, int, char*, char*);
void wait_message (int, int, char*, char*);
void message (int, int, char*, char*);
void clear_message (void);
void build_path (char*, char*, char*, int);
void restore_window(void);
extern char  dat_path[80];
extern char  plot_mic_energy_flag[4];

logical up_plot=FALSE,down_plot=FALSE;                    /* Plot pos. flags */

void start_plot()
{
 unsigned word_cnt;
 int i,start_word,stop_word;
 char filename[81],path2[80];
 FILE   *fileptr;          /* File pointer for word file "word.lst" */

 if(end_word > 16) end_word = 16;
 if((strcmp(plot_tmps,"YES") == 0) && (end_word > 11)) end_word = 11;
 strcpy(filename,"c:\\borlandc\\thesis\\word.lst");
 if ((fileptr = fopen(filename,"r"))  == NULL)
 {
  wait_message(0,7,"WARNING:  fopen of word.lst file failed! ",
  "Location:  start_plot() routine in plt_spc.c");
 }              /* Load the word array & create files */
 for (i=0;i<=end_word;i++) fgets(word_buffer[i],40,fileptr);
 fclose(fileptr);up_plot=FALSE;down_plot=FALSE;
 for(word_cnt=beg_word;word_cnt<=end_word;word_cnt++)
 {
  build_path(complete_file_name,drive,path,word_cnt);
  if (*plot_photo_time_flag == 'Y')
  {
   if ((up_plot==TRUE) && (down_plot==TRUE))
   {
       erase_plot();
       up_plot=FALSE;
       down_plot=FALSE;
   }
   if ((up_plot==FALSE) && (down_plot==FALSE))
```

```
        init_plot();
     if(strcmp("YES",plot_tmps) == 0)
     {
        build_path2(complete_file_name,drive,path,"p",word_cnt);
        plot_photo_tmps('p',word_cnt);
     }
     else
     {
        plot_time('p',word_cnt);       /* Plot photo time */
        if(*plot_photo_freq_flag == 'Y') plot_freq('p');  /* Plot photo freq*/
     }
     if ((up_plot==TRUE) && (down_plot==TRUE))
        if((print_flag[0] = getch()) == 'p') print_plot();
   }
   if (*plot_mic_time_flag == 'Y')
   {
     if ((up_plot==TRUE) && (down_plot==TRUE))
     {
        erase_plot();
        up_plot=FALSE;
        down_plot=FALSE;
     }
     if ((up_plot==FALSE) && (down_plot==FALSE))
        init_plot();
     if(strcmp("YES",plot_tmps) == 0)
     {
        build_path2(complete_file_name,drive,path,"m",word_cnt);
        plot_photo_tmps('m',word_cnt);
     }
     else if(plot_mic_energy_flag[0] == 'Y')
     {
        build_path(complete_file_name,drive,path,word_cnt);
        plot_mic_energy(word_cnt);
     }
     else
     {
        plot_time('m',word_cnt);       /* Plot mic time */
        if (*plot_mic_freq_flag == 'Y') plot_freq('m');        /* Plot mic freq*/
     }
     if ((up_plot==TRUE) && (down_plot==TRUE))
        if((print_flag[0] = getch()) == 'p') print_plot();
   }
  }
  if ((up_plot==TRUE) && (down_plot==FALSE))
    if((print_flag[0] = getch()) == 'p') print_plot();
```

```
 if ((up_plot==TRUE) || (down_plot==TRUE))
   erase_plot();
 return;
}


void plot_time(char flag,unsigned word_cnt)
{
 double max_x,max_y,min_x,min_y;
 float time_res=0,start_t=0,huge *y_buf;
 unsigned finish_flag,tol_num_plots,max_n;
 unsigned long buf_size,run_cnt,i,ii;
 int n;
 char file[40]="",pltpos[3],data_type[4]="";
 label_struct labels[16];

                        /* Open file to verify tol_num_plots */
 get_file_info(&tol_num_plots,&numbytes);
 /* where: tol_num_plots = number of words and/or sentences &  */
 /*         numbytes = window size used to record data */
 close(file_handle); /* "file_handle" is a global variable used by */
                        /* get_file_info to open files and leave open */
 if (end_mult > tol_num_plots) end_mult = tol_num_plots;
 if (flag == 'p')                    /* Plot photo data */
 {
  max_y = 5.0;min_y = 0.0;
  time_res = 1.0/2500.0;       /* 2500 = A/D sampling freq. */
  strcpy(&file[0],"c:\\borlandc\\thesis\\photmplt.dat");
  strcpy(data_type,"photo");
 }
 else                      /* Plot Mic time data */
 {
  max_y = 2.5;min_y = -2.5;
  time_res = 1.0/25000.0;     /* 25000 = A/D sampling freq. */
  strcpy(&file[0],"c:\\borlandc\\thesis\\mictmplt.dat");
  strcpy(data_type,"mic");
 }
 min_x = start_time;max_x = stop_time;
 if (up_plot==FALSE)
 {
  up_plot=TRUE;
  strcpy(&pltpos[0],"u");
 }
 else if (down_plot==FALSE)
 {
  strcpy(&pltpos[0],"l");
  down_plot=TRUE;
 }
 set_up_plt(file,max_x,min_x,max_y,min_y,pltpos);
 for (i=beg_mult;i<=end_mult;i++)
 {
  start_t = start_time;
  finish_flag = TRUE;  /* Tells convert_data that this is the 1st run */
  do
  {
   y_buf = convert_data(&finish_flag,i,data_type,&buf_size);
   if (flag == 'p') for(ii=0;ii<buf_size;ii++) y_buf[ii] += 2.5;
   plot(y_buf,time_res,start_t,buf_size,i-1);
   farfree((void *)y_buf);start_t += buf_size * time_res;
   if (finish_flag == TRUE) break;
  }
  while (finish_flag != TRUE);
 }
 strcpy(labels[0].l,word_buffer[word_cnt]);
 strcpy(labels[0].t,"i");
 labels[0].lt = 0;
 label_plot(labels,1,pltpos);
 return;
}

void plot_freq(char flag)
{
 double max_x=0,max_y=0; /* Max x and y values    */
 double min_x=0, min_y=0;              /* Min x and y values  */
 double min_volts,absolute_min_volts;
 float x_beg,mag,freq_res,fs,new_fs;
 float stop_t,start_t,volts[1],max_amp;
 float huge *in_buf,huge *real_arr,huge *img_arr;
 int INV,skip_flag;
 char file[40]="",pltpos[3],data_type[4],plot_info[50],p_buffer[12];
 unsigned long i,real2_size,new_buf_size,max_ints;
 unsigned long num_ints,max_arr_size,count,max_n,beg_i,end_i;
 unsigned long stop_count;
 unsigned tol_num_plots,run_cnt,finish_flag,M,skip_int;
 label_struct labels[16];
 logical neg_volts_flag,run_flag;
 FILE *tmpfile;

                    /* Open file to verify tol_num_plots */
 get_file_info(&tol_num_plots,&numbytes);
```

```
/* where: tol_num_runs = number of words and/or sentences &  */
/*        numbytes = window size used to record data */
close(file_handle); /* "file_handle" is a global variable used by */
                    /* get_file_info to open files and leave open */
if (end_mult > tol_num_plots) end_mult = tol_num_plots;
if (flag == 'p')        /* Plot photo freq */
{
 strcpy(file,"c:\\borlandc\\thesis\\phofrplt.dat");
 min_x = start_photo_freq;max_x = end_photo_freq;
 min_y = 0;max_y = max_photo_amp;
 strcpy(data_type,"p");
 fs = 2500.0;  /* Sampling freq in words/sec */
}
else                    /* Plot Mic freq data */
{
 strcpy(file,"c:\\borlandc\\thesis\\micfrplt.dat");
 min_x = start_mic_freq;max_x = end_mic_freq;
 min_y = 0;max_y = max_mic_amp;
 strcpy(data_type,"mic");
 fs = 25000.0;          /* Sampling freq in words/sec */
}
stop_t = stop_time;
start_t = start_time;
if (up_plot==FALSE)
{
 up_plot=TRUE;
 strcpy(&pltpos[0],"u");
}
else if (down_plot==FALSE)
{
 strcpy(&pltpos[0],"l");
 down_plot=TRUE;
}
beg_i = fs*start_t;end_i = fs*stop_t;
num_ints = end_i-beg_i;
for (run_cnt=beg_mult;run_cnt<=end_mult;run_cnt++)
{               /* Use the following temp file to process data */
 if ((tmpfile = fopen("c:\\borlandc\\thesis\\tmp.dat","wb+")) == NULL)
 {
  printf("ERROR(1):  data file open failed!\n");
  printf("Location:  plot_freq routine in plt_spc.c.\n");
  perror(" ");
  exit(1);
 }
 finish_flag = TRUE;        /* Tells convert_data 1st run */
 stop_count = 0;
 do
 {
  in_buf = convert_data(&finish_flag,run_cnt,data_type,&buf_size);
  fwrite((void *)in_buf,sizeof(float),buf_size,tmpfile);
  farfree((void *)in_buf);
  stop_count += buf_size;
 }
 while(finish_flag==FALSE);
 max_ints = farcoreleft()/4/2;
 if(num_ints<max_ints) M = floor(log10(num_ints)/log10(2));
 else M = floor(log10(max_ints)/log10(2));
 if(M>13) M = 13;  /* An upper limit FFT processing time constraint */
 max_ints = pow(2,M);
 skip_int = floor((float)num_ints/(float)max_ints);
 if ((real_arr = farcalloc(max_ints,sizeof(float))) == NULL)
 {
  printf("ERROR(2):  real array memory allocation failed.\n");
  printf("num_ints = %lu \n",num_ints);
  perror("");
  printf("Location: plot_freq() routine in plt_spc.c.\n");
  exit(1);
 }
 if ((img_arr = farcalloc(max_ints,sizeof(float))) == NULL)
 {
  printf("ERROR(3):  img array memory allocation failed.\n");
  printf("num_ints = %lu \n",num_ints);
  perror("");
  printf("Location: plot_freq() routine in plt_spc.c.\n");
  farfree((void *)real_arr);
  exit(1);
 }
 if (fseek(tmpfile,0L,SEEK_SET) != 0)
 {
  printf("ERROR(4):  data file fseek failed!\n");
  printf("Location: plot_freq() routine in plt_spc.c.\n");
  perror("");
  exit(1);
 }
 count = 0;skip_flag = skip_int + 1;
 for(i=0;i<stop_count;i++)
 {
  fread((void *)volts,sizeof(float),1L,tmpfile);
  if(skip_flag > skip_int)  /* Throw away data to meet end points */
  {
```

```
                real_arr[count] = volts[0];
                count++;
                skip_flag = 0;
            }
            skip_flag++;
          }
          fclose(tmpfile);      /* Calc new fs based on skipped data */
          new_fs=(float)count/(stop_t-start_t);
          INV = FALSE;
          fft2(real_arr,img_arr,M,INV);
          max_amp = -1.0;
          for (i=0;i<max_ints;i++)
          {
            real_arr[i] = log10(sqrt(pow(real_arr[i],2)+pow(img_arr[i],2)));
            max_amp = max(real_arr[i],max_amp);
          }
          farfree((void *)img_arr);
          freq_res = new_fs/(float)max_ints;
          if(run_cnt == beg_mult)
          {
            if(max_y<max_amp) max_y = max_amp;
            if(max_x>freq_res*max_ints/2.0) max_x = freq_res*max_ints/2.0;
            set_up_plt(file,max_x,min_x,max_y,min_y,pltpos);
          }
          plot(real_arr,freq_res,0.0,max_ints/2,run_cnt-1);
          farfree((void *)real_arr);
        }
        strcpy(plot_info,"Freq res. = ");
        gcvt(freq_res,5,p_buffer);
        strcat(plot_info,p_buffer);
        strcat(plot_info," Hz");
        strcpy(labels[0].l,plot_info);
        strcpy(labels[0].t,"i");
        labels[0].lt = 0;
        strcpy(plot_info,"Start time = ");
        gcvt(start_t,5,p_buffer);
        strcat(plot_info,p_buffer);
        strcat(plot_info," sec");
        strcpy(labels[1].l,plot_info);
        strcpy(labels[1].t,"i");
        labels[1].lt = 0;
        strcpy(plot_info,"Stop time = ");
        gcvt(stop_t,5,p_buffer);
        strcat(plot_info,p_buffer);
        strcat(plot_info," sec");
```

```
 strcpy(labels[2].l,plot_info);
 strcpy(labels[2].t,"i");
 labels[2].lt = 0;
 label_plot(labels,3,pltpos);
 return;
}


void plot_photo_tmps(char type_flag,unsigned word_cnt)
{
 double max_x,max_y,min_x,min_y;
 float huge *template_buf;
 unsigned numread,finish_flag,tol_num_plots;
 unsigned long buf_size,run_cnt,i,file_size;
 char file[40]="",pltpos[3];
 label_struct labels[16];
 FILE *file_ptr;

 if ((file_ptr = fopen(complete_file_name,"rb")) == NULL)
 {
  wait_message(0,7,"ERROR(1): data file open failed!",
  "Location: plot_photo_tmps routine in plt_spc.c");
  perror(" ");
  exit(1);
 }      /* Find file size in bytes */
if ((file_size = filelength(fileno(file_ptr))) == -1L)
{
  printf("ERROR(2): file size routine failed!\n");
  printf("Location: plot_photo_tmps routine in plt_spc.c\n");
  perror("");
  exit(1);
}              /* Convert file size to number of floats */
buf_size = file_size/4;
               /* Read choosen template_buf number */
if ((template_buf = farcalloc(buf_size+1,sizeof(float))) == NULL)
{
 printf("ERROR(3): template_buf buffer allocation failed!\n");
 printf("Location: plot_photo_tmps routine in plt_spc.c\n");
 perror("");
 exit(1);
}
numread=fread((void *)template_buf,sizeof(float),buf_size,file_ptr);
fclose(file_ptr);
max_y = -50.0;min_y = 0.0;min_x = 0;max_x = buf_size-1;
for(i=0;i<buf_size;i++) max_y = max(max_y,template_buf[i]);
```

```
 if(type_flag == 'p') strcpy(&file[0],"c:\\borlandc\\thesis\\pltptemp.dat");
 else strcpy(&file[0],"c:\\borlandc\\thesis\\pltmtemp.dat");
 if (up_plot==FALSE)
 {
  up_plot=TRUE;
  strcpy(&pltpos[0],"u");
 }
 else if (down_plot==FALSE)
 {
  strcpy(&pltpos[0],"l");
  down_plot=TRUE;
 }
 set_up_plt(file,max_x,min_x,max_y,min_y,pltpos);
 plot(template_buf,1,0,buf_size,1);
 farfree((void *)template_buf);
 strcpy(labels[0].l,word_buffer[word_cnt]);
 strcpy(labels[0].t,"i");
 labels[0].lt = 0;
 label_plot(labels,1,pltpos);
 return;
}

void plot_mic_energy(unsigned word_cnt)
{
 float *energy(float *time_inv,unsigned long *n,unsigned mult_num);
 float *filter_data(float*,unsigned long,unsigned long*);
 double max_x,max_y,min_x,min_y;
 float huge *energy_buf,time_inv;
 unsigned i,mult_num,tol_num_plots,max_n;
 unsigned long n;
 char file[40]="",pltpos[3];
 label_struct labels[16];

 get_file_info(&tol_num_plots,&numbytes);
 /* where: tol_num_plots = number of words and/or sentences &  */
 /* numbytes = window size used to record data */
 close(file_handle); /* "file_handle" is a global variable used by */
                     /* get_file_info to open files and leave open */
 if (end_mult > tol_num_plots) end_mult = tol_num_plots;
 if(up_plot==FALSE)
 {
  up_plot=TRUE;
  strcpy(&pltpos[0],"u");
 }
 else if (down_plot==FALSE)
```

```
{
 strcpy(&pltpos[0],"l");
 down_plot=TRUE;
}
strcpy(&file[0],"c:\\borlandc\\thesis\\energy.dat");
max_y = -5000;min_y = 0.0;
min_x = start_time;max_x = stop_time;
for(mult_num=beg_mult;mult_num<=end_mult;mult_num++)
{
 energy_buf = energy(&time_inv,&n,mult_num);
 energy_buf = filter_data(energy_buf,n,&n);
 energy_buf = filter_data(energy_buf,n,&n);
 for(i=0;i<n;i++) max_y = max(max_y,energy_buf[i]);
 if(mult_num==beg_mult) set_up_plt(file,max_x,min_x,max_y,min_y,pltpos);
 plot(energy_buf,time_inv,0,n,mult_num-1);
 farfree((void *)energy_buf);
}
strcpy(labels[0].l,word_buffer[word_cnt]);
strcpy(labels[0].t,"i");
labels[0].lt = 0;
label_plot(labels,1,pltpos);
return;
}
```

"Template.c" Source Program

```
/*---------------------------------------------------------------

Program contains speech processing functions used by main.  Creates
speech audio and photo templates.

Program:  template.c
Programmer:  Patrick T. Marshall
Date:  2/25/91
Organization:  WRDC/AAWP-2,
               WPAFB, OH 45433
Phone:  (513) 255-2471

-----------------------------------------------------------------------*/
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\plot.h"

extern unsigned long buf_size;              /* Buffer size */
extern unsigned long numbytes;              /* Total number of bytes to record */
extern int file_handle,window_width,max_step;
extern unsigned seg;
extern unsigned long tol_mem_avail,buf_count;
extern char word_buffer[12][20];       /* Used to store word name strings */
extern char  drive[MAXDRIVE],file_name[MAXFILE],ext[MAXEXT];
extern char  dat_path[80],tmp_path[80],ave_template_flag[4];
extern char  complete_file_name[MAXPATH],dir[MAXDIR];
extern char  ir_flag[4],mic_flag[4],print_flag[4];
extern int   beg_word,end_word;
extern int   beg_mult,end_mult;
extern int   num_words;
extern float start_time,stop_time;

/* Function prototypes */
void set_up_plt(char[],double,double,double,double,char[]);
void plot(float huge *,float,float,unsigned long,int);
void plot2(float huge*,float huge*,unsigned,int);
void label_plot(label_struct[],int,char[]);
void print_plot(void);
void erase_plot(void);
void init_plot();
float *iso_pho_env(float huge*,unsigned long,unsigned *);
float *iso_mic_fft(unsigned,unsigned*);
float *normalize(unsigned,float huge*,float *,unsigned *);
float huge *convert_data(unsigned *,unsigned,char[],unsigned long *);
get_file_info(unsigned*,unsigned long *);
char ask_question (int, int, char*, char*);
void wait_message (int, int, char*, char*);
void message (int, int, char*, char*);
void clear_message (void);
void build_path (char*, char*, char*, int);
void build_path2 (char*, char*, char*, char*, int);
void restore_window(void);
float min_distance(float[],float[],unsigned);

char path2[80];
unsigned word_cnt;

/*************************************************************************
  The following creates word templates and stores results to a file.
*************************************************************************/
void create_templates()
{
 void sensor_fusion_segmentor(char[],char);
 void find_template(char[],char,char[]);
 int i;
 char tmp_file_name[50];
 char filename[81];
 FILE   *fileptr;          /* File pointer for word file "word.lst" */

 strcpy(filename,"c:\\borlandc\\thesis\\isoword.lst");
 if(end_word > 11) end_word = 11;
 if ((fileptr = fopen(filename,"r"))  == NULL)
 {
  wait_message(0,7,"ERROR(1):  fopen of word.lst file failed! ",
    "Location:  create_templates() routine in TEMPLATE.C");
    exit(0);
  }            /* Load the word array */
  for (i=0;i<=end_word;i++) fgets(word_buffer[i],20,fileptr);
  fclose(fileptr);
  for(word_cnt=beg_word;word_cnt<=end_word;word_cnt++)
  {
    build_path(complete_file_name,drive,dat_path,word_cnt);
    sensor_fusion_segmentor(complete_file_name,'i');
/*
    build_path2(tmp_file_name,drive,tmp_path,"p",word_cnt);
    printf("Data file: %s\n",complete_file_name);
    printf("Template file: %s\n",tmp_file_name);
    find_template(complete_file_name,'p',tmp_file_name);
*/
    build_path2(tmp_file_name,drive,tmp_path,"m",word_cnt);
    printf("Data file: %s\n",complete_file_name);
    printf("Template file: %s\n",tmp_file_name);
    find_template(complete_file_name,'m',tmp_file_name);
  }
  return;
}

/**************************************************************************
  The following is called from create_templates() routine.
  Its purpose is to find the best template for each word, for each person.
  The DWT algorithm is used to determine which template is the best for
  each word.
**************************************************************************/
void find_template(char filename[],char data_type,char template_file_name[])
{
  unsigned *warp(float*,unsigned,float*,unsigned,float*,unsigned*);
  float *process_pho_data(char[],float,float,unsigned,unsigned*);
  float *process_mic_data(char[],float,float,unsigned,unsigned*);
  float *find_words(char[],unsigned*,char,unsigned);
  void dpfunc(unsigned,unsigned*,unsigned*);
  float *filter_data(float*,unsigned,unsigned*);
  float *proc_dat,*ave_buf,*word_times;
  float tcost,max_y,fs,*template,*test_pat,*warped_buf;
  float min_tol_cost,tol_cost,beg_time,end_time;
  unsigned long buf_size;
  unsigned template_num,max_mult_num,k,peak_n,word_len,word_dis;
  unsigned i,ii,jj,count=0,*map,n=0,l,beg_i,end_i;
  unsigned num_words_found,template_len;
  char file[50];
  label_struct labels[16];
  FILE *fptr[6],*outfile,*timeptr;

  strcpy(complete_file_name,filename);
  get_file_info(&max_mult_num,&numbytes);
  /* where: max_mult_num = number of words and/or sentences &  */
  /*        numbytes = window size used to record data */
  close(file_handle); /* "file_handle" is a global variable used by */
                      /* get_file_info to open files and leave open */
  if (end_mult > max_mult_num) end_mult = max_mult_num;
  beg_i=beg_mult;end_i=end_mult;
  if(data_type == 'p') fs = 2500.0;
  else fs = 25000.0;
  system("del *.$$$");
/*
  strcpy(&file[0],"c:\\borlandc\\thesis\\photmplt.dat");
  init_plot();
  set_up_plt(file,110,0,0.3,0,"u");
*/
  for (i=beg_i;i<=end_i;i++)  /* Create temporary word files */
  {
   if((fptr[i] = tmpfile()) == NULL)
   {
    printf("ERROR(2):  temporary file open failed!\n");
    printf("Location:  find_template() routine in template.c\n");
    perror("");exit(0);
   }
   if(data_type == 'p')
    build_path(file,drive,"\\borlandc\\thesis\\phowrdtm",i);
   else build_path(file,drive,"\\borlandc\\thesis\\micwrdtm",i);
   if((timeptr = fopen(file,"rb")) == NULL)
   {                    /* Pointer to word endpoint time file */
    wait_message(0,7,"ERROR(3):  data file open failed!",
        "Location:  find_template() routine in template.c");
    perror(" ");exit(0);
   }
   fread(&num_words_found,sizeof(unsigned),1L,timeptr);
   fread(&beg_time,sizeof(float),1L,timeptr);
   fread(&end_time,sizeof(float),1L,timeptr);
   fclose(timeptr);
   if(data_type == 'p') template = process_pho_data(complete_file_name,
                                    beg_time,end_time,i,&n);
   else template = process_mic_data(complete_file_name,
                                    beg_time,end_time,i,&n);
   template = normalize(n,template,&max_y,&peak_n);
```

```
/*
 plot(template,1.0,0.0,n,2);
*/
   if(data_type == 'p') template = filter_data(template,n,&n);
   fwrite((void *)template,sizeof(float),n,fptr[i]);
   farfree((void *)template);
 }
 for (i=beg_i;i<=end_i;i++) rewind(fptr[i]);
 min_tol_cost = 1e+5;tol_cost = 0;
 for (i=beg_i;i<=end_i;i++)
 {       /* Find min. cost multiple word file to be DWT template */
  n = (unsigned)filelength(fileno(fptr[i]));
  n /= sizeof(float);
  if((template = farcalloc(n+1,sizeof(float))) == NULL)
  {     /* Buffer to hold averaged data */
   printf("ERROR(3):  buffer allocation failed!\n");
   printf("Location:  find_template() routine in template.c\n");
   perror("");exit(0);
  }
  fread((void *)template,sizeof(float),n,fptr[i]);
  rewind(fptr[i]);
  template_len = n;
  for (ii=beg_i;ii<=end_i;ii++)
  {
   if(ii == i)
      if(ii == end_i) break;
      else ii++;
   n = (unsigned)filelength(fileno(fptr[ii]));
   n /= sizeof(float);
   if((test_pat = farcalloc(n+1,sizeof(float))) == NULL)
   {  /* Buffer to hold averaged data */
      printf("ERROR(4):  buffer allocation failed!\n");
      printf("Location:  find_template() routine in template.c\n");
      perror("");exit(0);
   }
   fread((void *)test_pat,sizeof(float),n,fptr[ii]);
   rewind(fptr[ii]);
   if(data_type == 'p')
   {
      max_step = 2;
      window_width = 1;
   }
   else
   {
      max_step = 2;
```

```
        window_width = 4;
      }
      map = warp(test_pat,n-1,template,template_len-1,&tcost,&k);
      tol_cost += tcost;
      farfree((void *)test_pat);farfree((void *)map);
    }
    min_tol_cost = min(min_tol_cost,tol_cost);
    if(min_tol_cost == tol_cost) template_num = i;
    farfree((void *)template);tol_cost = 0;
  }
  for (i=beg_i;i<=end_i;i++) rewind(fptr[i]);
  n = (unsigned)filelength(fileno(fptr[template_num]));
  n /= sizeof(float);
  if((template = farcalloc(n+1,sizeof(float))) == NULL)
  {       /* Buffer to hold averaged data */
   printf("ERROR(5):  buffer allocation failed!\n");
   printf("Location:  find_template() routine in template.c\n");
   perror("");exit(0);
  }
  fread(&template[0],sizeof(float),n,fptr[template_num]);
  if((outfile = fopen(template_file_name,"wb+")) == NULL)
  {
   wait_message(0,7,"ERROR(6):  data file open failed!",
         "Location:  find_template() routine in template.c");
   perror(" ");exit(0);
  }
  fwrite((void *)template,sizeof(float),n,outfile);
  fclose(outfile);farfree((void *)template);
  for (i=beg_i;i<=end_i;i++) fclose(fptr[i]);
  return;
}


/***************************************************************************
 The following finds the best template amoung existing templates.
 A list of template names are in an ASCII file: "template.lst".
 The DWT algorithm is then used to determine which template is the
 best for each word.
***************************************************************************/
void find_best_templates()
{
 unsigned *warp(float*,unsigned,float*,unsigned,float*,unsigned*);
 void dpfunc(unsigned,unsigned*,unsigned*);
 float *ave_buf,*template,*test_pat,*warped_buf;
float tcost,min_tol_cost,tol_cost;
long filesize=0,temp_filesize=0;
unsigned template_num=0;
unsigned numwrite,numread,max_mult_num,k,word_dis;
unsigned i,ii,iii,j,*map;
unsigned l,num_file_names=0;
char template_file_names[10][15],filename[81],data_type[2];
label_struct labels[16];
FILE   *fileptr;          /* File pointer */

strcpy(filename,"c:\\borlandc\\thesis\\isoword.lst");
if(end_word > 11) end_word = 11;
if ((fileptr = fopen(filename,"r"))  == NULL)
{
 wait_message(0,7,"ERROR(1):  fopen of word.lst file failed! ",
 "Location:  find_best_templates() routine in TEMPLATE.C");
 exit(0);
}                                  /* Load word list */
for (i=beg_word;i<=end_word;i++) fgets(word_buffer[i],20,fileptr);
fclose(fileptr);
strcpy(filename,"c:\\borlandc\\thesis\\template.lst");
if ((fileptr = fopen(filename,"r"))  == NULL)
{
 wait_message(0,7,"ERROR(2):  fopen of tempate.lst file failed! ",
 "Location:  find_best_templates() routine in TEMPLATE.C");
 exit(0);
}                        /* Load template file names */
num_file_names = 0;
/*****************************************************************/
/* Check to make sure the following does not add an extra number */
/*****************************************************************/
while(fscanf(fileptr,"%s",template_file_names[num_file_names]) != EOF)
 num_file_names++;
fclose(fileptr);
strcpy(data_type,"p");          /* Do photo first */
do
{
 for (i=beg_word;i<=end_word;i++)
 {
  min_tol_cost = 1e+5;tol_cost = 0;
  for (ii=0;ii<num_file_names;ii++)
  { /* Find min. cost multiple word file to be DWT template */
     build_path2(filename,drive,template_file_names[ii],data_type,i);
     printf("Reference file: %s\n",filename);
     if ((fileptr = fopen(filename,"rb"))  == NULL)
```

```
{
 wait_message(0,7,"ERROR(3): fopen of template file failed! ",
 "Location: find_best_templates() routine in TEMPLATE.C");
 perror("");exit(0);
}
temp_filesize = filelength(fileno(fileptr));
temp_filesize /= sizeof(float); /* Convert bytes to floats */
if ((template = farcalloc(temp_filesize+1,sizeof(float))) == NULL)
{
 printf("ERROR(4): template buffer allocation failed!\n");
 printf("Location: find_best_templates() routine in template.c\n");
 perror("");exit(0);
}              /* Open & read file (template) for DTW */
numread=fread((void *)template,sizeof(float),temp_filesize,fileptr);
fclose(fileptr);
for(iii=0;iii<num_file_names;iii++)
{
 if(iii == ii)
  if(iii == (num_file_names-1)) break;
  else iii++;
 build_path2(filename,drive,template_file_names[iii],data_type,i);
 if ((fileptr = fopen(filename,"rb"))  == NULL)
 {
  wait_message(0,7,"ERROR(5): fopen of file failed! ",
  "Location: find_best_templates() routine in TEMPLATE.C");
  perror("");exit(0);
 }
 filesize = filelength(fileno(fileptr));
 filesize /= sizeof(float);       /* Convert bytes to floats */
 if ((test_pat = farcalloc(filesize,sizeof(float))) == NULL)
 {
  printf("ERROR(6): test_pat buffer allocation failed!\n");
  printf("Location: find_best_templates() routine in template.c\n");
  perror("");exit(0);
 }
 numread=fread((void *)test_pat,sizeof(float),filesize,fileptr);
 if(data_type == 'p')
 {
  max_step = 2;
  window_width = 1;
 }
 else
 {
  max_step = 2;
  window_width = 4;
```

```
          }
          map = warp(test_pat,filesize-1,template,temp_filesize-1,&tcost,&k);
          tol_cost += tcost;
          farfree((void *)map);
          farfree((void *)test_pat);fclose(fileptr);
        }
        min_tol_cost = min(min_tol_cost,tol_cost);
        if(min_tol_cost == tol_cost) template_num = ii;
        farfree((void *)template);tol_cost = 0;
      }
      build_path2(filename,drive,template_file_names[template_num],data_type,i);
      printf("Choosen template = %s\n",template_file_names[template_num]);
      if ((fileptr = fopen(filename,"rb"))  == NULL)
      {
        wait_message(0,7,"ERROR(7):  fopen of template file failed! ",
        "Location:  find_best_templates() routine in TEMPLATE.C");
        perror("");exit(0);
      }
      temp_filesize = filelength(fileno(fileptr));
      temp_filesize /= sizeof(float); /* Convert bytes to floats */
      if ((template = farcalloc(temp_filesize+1,sizeof(float))) == NULL)
      {
        printf("ERROR(8):  template buffer allocation failed!\n");
        printf("Location:  find_best_templates() routine in template.c\n");
        perror("");exit(0);
      }
      numread=fread((void *)template,sizeof(float),temp_filesize,fileptr);
      fclose(fileptr);
      build_path2(filename,drive,"besttemp",data_type,i);
      if ((fileptr = fopen(filename,"wb"))  == NULL)
      {
        wait_message(0,7,"ERROR(9):  fopen of best template file failed! ",
        "Location:  find_best_templates() routine in TEMPLATE.C");
        perror("");exit(0);
      }
      numwrite=fwrite((void *)template,sizeof(float),temp_filesize,fileptr);
      farfree((void *)template);fclose(fileptr);
    }
    if(data_type[0] == 'm') strcpy(data_type,"f");
    if(data_type[0] == 'p') strcpy(data_type,"m");
  }
  while(data_type[0] != 'f');
  return;
}
```

"Spch_rec.c" Source Program

```
/*-----------------------------------------------------------------

Program contains speech recognition algorithms.

Program: spch_rec.c
Programmer: Patrick T. Marshall
Date: 2/25/91
Organization: WRDC/AAWP-2,
              WPAFB, OH 45433
Phone: (513) 255-2471

-----------------------------------------------------------------------*/
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\plot.h"

extern unsigned long buf_size;              /* Buffer size */
extern unsigned long numbytes;              /* Total number of bytes to record */
extern int file_handle,window_width,max_step;
extern unsigned long tol_mem_avail,buf_count;
extern char word_buffer[19][41];       /* Used to store word name strings */
extern char drive[MAXDRIVE],file_name[MAXFILE],ext[MAXEXT];
extern char dat_path[80],tmp_path[80];
extern char complete_file_name[MAXPATH],dir[MAXDIR];
extern char ir_flag[4],mic_flag[4],print_flag[4];
extern int  beg_word,end_word,beg_mult,end_mult;
extern int  num_words,beg_template_num,end_template_num;
extern float start_time,stop_time;
```

```
float *normalize(unsigned,float huge*,float *,unsigned *);
float huge *convert_data(unsigned *,unsigned,char[],unsigned long *);
get_file_info(unsigned*,unsigned long *);
void wait_message (int, int, char*, char*);
void build_path (char*, char*, char*, int);
void build_path2 (char*, char*, char*, char*, int);

FILE *results_ptr;              /* For storing recognition results */


/***************************************************************************
 The following conducts speech recognition in boths modes (isolated or
 continous).
***************************************************************************/
void start_speech_rec(logical sensor_fusion_flag)
{
 void process_spch_rec(char[],char[],unsigned,unsigned,logical);
 float *find_words(char[],unsigned*,char,unsigned);

 unsigned i,ii;
 char filename[81],data_type[1];
 FILE *fileptr;           /* File pointer for word file "word.lst" */

 system("del c:\\borlandc\\thesis\\pho_fus.dat");
 strcpy(filename,"c:\\borlandc\\thesis\\results.dat");
 if ((results_ptr = fopen(filename,"a"))  == NULL)
 {
  wait_message(0,7,"WARNING:  fopen of results.dat file failed! ",
  "Location:  speech_rec() routine in spch_rec.c.c");
 }
 strcpy(filename,"c:\\borlandc\\thesis\\word.lst");
 strcpy(data_type,"p");                          /* Do photo first */
 if ((fileptr = fopen(filename,"r"))  == NULL)
 {
  wait_message(0,7,"WARNING:  fopen of word.lst file failed! ",
  "Location:  speech_rec() routine in spch_rec.c.c");
 }
 for (i=0;i<18;i++)    /* Load the word array & create the files. */
 {
  fgets(word_buffer[i],40,fileptr);
 }
 fclose(fileptr);
 do                                   /* Go through mic & photo */
 {
  if(sensor_fusion_flag == TRUE)
  {
   fprintf(results_ptr,"************* SENSOR FUSION RECOGNITION RESULTS
************\n");
   printf("************* SENSOR FUSION RECOGNITION RESULTS
************\n");
  }
  else if(data_type[0] == 'p')
  {
   fprintf(results_ptr,"*************** PHOTO RECOGNITION RESULTS
***************\n");
   printf("*************** PHOTO RECOGNITION RESULTS
***************\n");
  }
  else
  {
   fprintf(results_ptr,"************* MICROPHONE RECOGNITION RESULTS
************\n");
   printf("************* MICROPHONE RECOGNITION RESULTS
************\n");
  }
  for(i=beg_word;i<=end_word;i++)          /* Go through words */
  {
   build_path(complete_file_name,drive,dat_path,i); /* Creating word file */
   if(sensor_fusion_flag == TRUE)
   {
     if(end_word <= 11) sensor_fusion_segmentor(complete_file_name,'i');
     else sensor_fusion_segmentor(complete_file_name,'c');
   }
   for(ii=beg_mult;ii<=end_mult;ii++)   /* Go through word multiples */
   {
     printf("File:  %s \n",complete_file_name);
                              /* Find beg & end of words */
     fprintf(results_ptr,"File: %s \n",complete_file_name);
                              /* Find beg & end of words */
     process_spch_rec(complete_file_name,data_type,i,ii,sensor_fusion_flag);
   }
  }
  printf("\n");
  fprintf(results_ptr,"\n");
  if(data_type[0] == 'm') strcpy(data_type,"f"); /* (f)inished */
  else strcpy(data_type,"m");                    /* Do (m)ic next */
  if(sensor_fusion_flag == TRUE) strcpy(data_type,"f"); /* (f)inished */
 }
 while(data_type[0] != 'f');
 printf("Done with this run!\n");
 fclose(results_ptr);
 return;
}


/*************************************************************************
 The following is called from speech_rec() routine to process the speech
 and recognize words.  The word and word multiple are already processed by
 speech_rec() as "filename".  This routine will then find the number of
 words in filename's recorded speech and compare it to word templates.

 Inputs:
  filename - complete file name (i.e. contains complete path & drive).
  mult_num - word multiple number
  data_type - Type of data working with ("p" - photo or "m" for mic.)

 Outputs:
  none

 Global:
  complete_file_name - filename to be processed (includes path & drive)
  start_time & stop_time - used in convert_data()
*************************************************************************/
void process_spch_rec(char filename[],char data_type[],unsigned word_num,
                        unsigned mult_num,logical sensor_fusion_flag)
{
 unsigned *warp(float*,unsigned,float*,unsigned,float*,unsigned*);
 void dpfunc(unsigned,unsigned*,unsigned*);
 float *process_pho_data(char[],float,float,unsigned,unsigned*);
 float *process_mic_data(char[],float,float,unsigned,unsigned*);
 float min_distance(float[],float[],unsigned);
 float *linear_warp(float[],unsigned,float[],unsigned);
 void plot2(float huge*,float huge*,unsigned,int);
 float *find_words(char[],unsigned*,char,unsigned);

 float *word_buf,min_cost=-5000,*word_times,pho_cost;
 float tcost,max_y,max_y1;
 float *template_buf,beg_time,end_time;
 unsigned long temp_file_size;
 unsigned max_mult_num,k,max_n,i,ii,iii,*map,n=0,recognized_word;
 unsigned num_words_found,photo_word;
 char tmp_file_name[50],file[31]="",pltpos[2];
 FILE *template_file_ptr,*timeptr,*phoptr;

 strcpy(complete_file_name,filename);
```

```
get_file_info(&max_mult_num,&numbytes);
/* where: max_mult_num = number of words and/or sentences & */
/*        numbytes = window size used to record data */
close(file_handle); /* "file_handle" is a global variable used by */
                    /* get_file_info to open files and leave open */
start_time = 0;stop_time = numbytes/22.0/2500.0;
/* Create word file that will store processed words found in cont. sig. */
if (data_type[0] == 'p') strcpy(&file[0],"c:\\borlandc\\thesis\\photemp.dat");
else strcpy(&file[0],"c:\\borlandc\\thesis\\mictemp.dat");
if(sensor_fusion_flag == TRUE)
{
 build_path(file,drive,"\\borlandc\\thesis\\micwrdtm",mult_num);
 if((timeptr = fopen(file,"rb")) == NULL)
 {                    /* Pointer to word endpoint time file */
  printf("ERROR(1b):  data file open failed!\n");
  printf("Location:  process_spch_rec() routine in spch_rec.c\n");
  perror(" ");exit(0);
 }
 fread(&num_words_found,sizeof(unsigned),1L,timeptr);
 if((word_times = farcalloc(4*num_words_found,sizeof(float))) == NULL)
 {
  printf("ERROR(2):  template buffer allocation failed!\n");
  printf("Location:  process_spch_rec() routine in spch_rec.c\n");
  perror("");exit(0);
 }
 for(i=0;i<4*num_words_found;i+=4)  /* Read word time bondaries */
 {
  fread(&word_times[i],sizeof(float),1L,timeptr); /* beg. of word */
  fread(&word_times[i+3],sizeof(float),1L,timeptr); /* end of word */
 }
 fclose(timeptr);
}
else
{
 word_times = find_words(complete_file_name,&num_words_found,data_type[0],
                    mult_num);
 if(end_word < 12)
 {
  word_times[3] = word_times[4*(num_words_found-1)+3];
  num_words_found = 1;
  if(word_times[3] <= 0.0)
  {
     word_times[0] = start_time;
     word_times[3] = stop_time;
  }
  }
 }
for(i=0;i<4*num_words_found;i+=4)  /* Now do speech recognition */
{
 again:                    /* For processing mic. word for sensor fusion */
 beg_time = word_times[i];end_time = word_times[i+3];
 if (data_type[0] == 'p')
   word_buf = process_pho_data(filename,beg_time,end_time,mult_num,&n);
 else word_buf = process_mic_data(filename,beg_time,end_time,mult_num,&n);
 word_buf = normalize(n,word_buf,&max_y1,&max_n);
 min_cost = 5000;
 for(ii=beg_template_num;ii<=end_template_num;ii++)
 {                                  /* Loop for templates in lib. */
  if (data_type[0] == 'p')          /* Create photo template file */
     build_path2(tmp_file_name,drive,tmp_path,"p",ii);
  else                                    /* Create mic template file */
     build_path2(tmp_file_name,drive,tmp_path,"m",ii);
  if ((template_file_ptr = fopen(tmp_file_name,"rb")) == NULL)
  {
     printf("ERROR(3):  template file does not exist!\n");
     printf("Location:  process_spch_rec() routine in spch_rec.c\n");
     perror("");exit(0);
  }                                       /* Get file size */
  temp_file_size = filelength(fileno(template_file_ptr));
  temp_file_size /= 4;              /* Convert file size to float */
  rewind(template_file_ptr);        /* Move file pointer to beg. of file */
  if((template_buf = farcalloc(temp_file_size+1,sizeof(float))) == NULL)
  {
     printf("ERROR(4):  template buffer allocation failed!\n");
     printf("Location:  process_spch_rec() routine in spch_rec.c\n");
     perror("");exit(0);
  }
  fread((void *)template_buf,sizeof(float),temp_file_size,template_file_ptr);
  if(data_type[0] == 'p')
  {
     max_step = 2;
     window_width = 2;
  }
  else
  {
     max_step = 2;
     window_width = 4;
  }
  map = warp(word_buf,n-1,template_buf,temp_file_size-1,&tcost,&k);
  if(ii == beg_template_num) min_cost = tcost;
```

```
      min_cost = min(min_cost,tcost);
      if(min_cost == tcost) recognized_word = ii;
      fclose(template_file_ptr);
      farfree((void *)map);
      farfree((void *)template_buf);
     }
    farfree((void *)word_buf);
    if(sensor_fusion_flag == TRUE)
    {
     if(data_type[0] == 'p')
     {
        photo_word = recognized_word;
        pho_cost = min_cost;
        data_type[0] = 'm';
        goto again;
     }
     if((min_cost > 0.17) && (pho_cost < 0.005)) /* Use photo word */
     {
        recognized_word = photo_word;
        printf("Picked photo word!!!\n");
        fprintf(results_ptr,"Picked photo word!!!\n");
     }
     if((min_cost > 0.6) || (pho_cost < 0.0001)) /* Use photo word */
     {
        recognized_word = photo_word;
        printf("Picked photo word!!!\n");
        fprintf(results_ptr,"Picked photo word!!!\n");
     }
    data_type[0] = 'p';           /* Reset to photo word */
   }
   printf("From %f sec to %f sec =
%s\n",beg_time,end_time,word_buffer[recognized_word]);
   fprintf(results_ptr,"Time = %f sec to %f sec Word = %s",beg_time,end_time,
                        word_buffer[recognized_word]);
  }
  farfree((void *)word_times);
  return;
}
```

"DMA.c" Source Program

```
/************************************************************************/
/* dma.c -> Contains subroutines used for dma data acquisition
 *
 * The calling routine must declare the variables in the "extern" list
 * below, and the reset_irq() function. Communication from the main
 * program to the subroutines is mostly through these global variables.
 * The calling routine must give values to dma_chan, dma_irq and
 * buf_size, then call alloc_dma_buf(), dma_setup(), and start_dma().
 * As each dma buffer fills up, the interupt service routine calls
 * start_dma() on the next buffer. The calling routine can wait for
 * buf_index to change, then process data pointed to by curr_buf. Cleanup
 * is done by dma_finish(),which is called automatically when the program
 * exits.
 *
 * Compiler: Borland's Turbo C
 *          Set /Gs switch to remove stack probes (a necessity for
 *          any function called at interrupt state!)
 *
 * Written by Tom Nolan - 11/3/89
 * Modified by Pat Marshall - 9/17/90
 */
/************************************************************************/

#include <bios.h>
#include <time.h>
#include <conio.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <c:\borlandc\thesis\logical.h>
#pragma check_stack(off)

#define DMA0_BASE       0x00  /* Address of dma controller (chan 0-3) */
#define DMA1_BASE       0xC0  /* Address of dma controller (chan 4-7) */

/* Interrupt Contoller Definitions */
```

```
#define INTA00              0x20 /* Base address of int ctrlr */
#define INTA01              0x21 /* Address of int ctrlr 2nd reg */
#define EOI         0x20 /* Code for non-specific end-of-int */

/* Macros for extracting bytes from 20-bit addresses */
#define LSB(x)   *((unsigned char *) &x)
#define MSB(x)   *(((unsigned char *) &x) +1)
#define PAGE(x)  *(((unsigned char *) &x) +2)

typedef struct
        {
        unsigned char far *p; /* Pointers to buffers */
        unsigned long a;      /* Address of buffer */
        unsigned s;           /* Size of buffer */
        } buf_struc;

extern buf_struc dma_buffers[];              /* DMA buffers */
extern unsigned long numbytes;
extern char far *curr_buf;             /* Pointer to current buffer */
extern unsigned long buf_size;               /* Buffer size */
extern int buf_index;           /* Index of current buffer */
extern int dma_irq;             /* Hardware int request line */
extern int dma_chan;            /* Hardware DMA channel number */
extern int file_handle;                /* File handle */
extern int lost_buffers;               /* Write errors */
extern int irq_flag;
extern int numbuffers;
extern int bufs_filled;
extern unsigned statreg,seg;
extern unsigned long max;

/* Variables - placed in static storage to avoid excessive stack
 * usage in interrupt routines */

static union REGS  r;           /* General registers */
static struct SREGS s;          /* Segment registers */
static int sel;                 /* DMA channel select bits */
static int basereg;             /* DMA controller base address register */
static int cntreg;              /* DMA controller count register */
static int maskreg;             /* DMA controller mask register */
static int modereg;             /* DMA controller mode register */
static int pagereg;             /* DMA page address register */
static int page_tbl[] =         /* Table of page register addresses */
  { 0x87, 0x83, 0x81, 0x82,  /* for dma channels 0, 1, 2, 3,    */
    0x8f, 0x8b, 0x89, 0x8a }; /*                 4, 5, 6, 6, 7, */
```

```
char far *dos_crit_addr;         /* Address of DOS critical section flag */

static void                      /* Space for saved int vector contents */
  (interrupt far *dma_int_save) ();

int alloc_dma_buf(void);         /* Allocate dma buffers */
void intr_setup(void);           /* Set up interrupt operation */
void dma_setup(void);                   /* Set up dma operation */
void dma_finish(void);
void interrupt far dma_isr(void);
void start_dma(char far *,unsigned); /* Start a dma operation */
void init_brd(void);             /* Initialize A/D board */
void on_brd(void);               /* Turn A/D board on */


/*-------------------------------------------------------------------*/
int alloc_dma_buf()              /* Allocate dma buffers */
{
 unsigned buf;                   /* Temp variable for various paragraph */
                                 /* addresses                          */
 unsigned size;                  /* Buffer size in paragraphs */
 unsigned numpars,i;

 /* This routine allocates buffers that can be filled by dma.
  * The buffers are guaranteed to be aligned so that they do not cross
  * physical page boundaries. Before calling this routine, set the value
  * of numbytes to the required total number of bytes to be transfered.
  * Note that the maximum buffer size is 64K bytes. Also, the byte count
  * is converted to paragraphs, which are the units the DOS memory
  * allocation functions work with.
  * Buffer information:
  *     dma_buffers(i).p = pointer to ith buffer's 20-bit physical address
  *     dma_buffers(i).a = absolute address of ith buffer
  *     dma_buffers(i).s = size of ith buffer
  * The return is zero if the allocation succeded, non-zero (an MS-DOS
  * error code) otherwise.
  */

 max=allocmem(0xffff, &seg);              /* Get max paragraphs from dos */
 if(allocmem(max, &seg) != -1)     /* Now grab it all */
 {
  printf("Memory alloction failed!\n");
  return 1;
 }
 buf = seg;
```

```
    if ((numbytes >> 4) > max) numbytes = max << 4;
    numpars = numbytes >> 4;            /* Convert bytes to paragraphs */
    i = 0;numbytes = 0;
    while(numpars>8)
    {
     if(((but + numpars - 1) & 0xf000) != (buf & 0xf000))
     {                     /* If buffer crossesphys page boundary */
      dma_buffers[i].p = (char far *)    /* Convert buffer segment */
          ((long) buf << 16);            /* ... to far pointer for return */
      dma_buffers[i].a = (unsigned long) buf << 4;
      buf = (buf & 0xf000) +             /* ... adjust to next */
          0x1000;                              /* phys page */
     }
     else
     {
      dma_buffers[i].p = (char far *)    /* Convert buffer segment */
          ((long) buf << 16);            /* ... to far pointer for return */
      dma_buffers[i].a = (unsigned long) buf << 4;
      buf += numpars;                    /* Initial attempt at next buffer seg. */
     }
     dma_buffers[i].s = (unsigned) /* Convert buffer size to bytes */
      (buf << 4) - dma_buffers[i].a-16;
     numpars -= dma_buffers[i].s >> 4;
     numbytes += dma_buffers[i].s;
     i++;
    }
    numbuffers = i;
    i = numbuffers-1;
    size = ((dma_buffers[i].a+dma_buffers[i].s) >> 4) - seg;
                                         /* Compute actual size needed */
    if (setblock(seg,size) != -1)        /* return error if not enough */
    {
     printf("Setblock memory resizing failed!\n");
     return 0;
    }
}
/*------------------------------------------------------------------------*/
void intr_setup()                        /* Set up interrupt operation */
{
 /* Before calling this routine set the following variable:
  *     dma_irq  = interupt request number 0-7 (hardware dependent)
  */

  int intmsk;
```

```
    r.h.ah = 0x34;                 /* DOS "get critical flag addr" function */
    intdosx(&r,&r,&s);
    dos_crit_addr = (char far *)   /* Save its address so it can be tested */
      (((long) s.es << 16) | r.x.bx); /* ... as a far pointer */
    if(dma_irq < 8)                /* Save current contents of dma int vec */
    {
      dma_int_save = getvect(dma_irq + 8); /* For IRQ's 0 - 7 */
      setvect(dma_irq+8, dma_isr); /* Set up new int service routine */
    }
    else
    {
      dma_int_save = getvect(dma_irq + 104); /* For IRQ's 8 - 15 (AT only) */
      setvect(dma_irq+104, dma_isr); /* Set up new int service routine */
    }
    intmsk = inp(INTA01);          /* Get current interrupt enable mask */
    intmsk &= ~(1 << dma_irq);            /* Clear mask bit for dma interrupt */
    outp(INTA01, intmsk);          /* Output new mask, enabling interupt */
  }
  /*-------------------------------------------------------------------------*/
  void dma_setup()                     /* Set up dma operation */
  {
   /* Before calling this routine set the following variable:
    *     dma_chan = channel number (hardware dependent)
    */


    sel = dma_chan & 3;                       /* Isolate channel select bits */
    pagereg = page_tbl[dma_chan];      /* Locate corresponding page reg */
    if(dma_chan < 4)                   /* Setup depends on chan number */
    {
      basereg = DMA0_BASE + sel * 2;          /* Standard dma controller */
      cntreg  = basereg + 1;           /* Note that this controller */
      maskreg = DMA0_BASE + 10;               /* is addressed on "byte" */
      modereg = DMA0_BASE + 11;               /* boundaries */
      statreg = DMA0_BASE + 8;
    }
    else
    {
      basereg = DMA1_BASE + sel * 4;  /* Alternate dma controller (AT only) */
      cntreg =  basereg + 2;          /* Note that this controller */
      maskreg = DMA1_BASE + 20;       /* is addressed on "word" */
      modereg = DMA1_BASE + 22;       /* boundaries */
      statreg = DMA1_BASE + 16;
    }
  }
```

```
/*-----------------------------------------------------------------------*/
void dma_finish()
{
 int intmsk;

 freemem(seg);                  /* Free memory */
 init_brd();                       /* Initialize A/D board */
 intmsk = inp(INTA01);                  /* Get current interupt enable mask */
 intmsk |= (1 << dma_irq);      /* Set mask bit for dma interrupt */
 outp(INTA01, intmsk);                  /* Output new mask, disabling interupt */
 setvect(dma_irq+8, dma_int_save); /* Restore old vector contents */
}

/*-----------------------------------------------------------------------*/
void interrupt far dma_isr()
{

 /* This routine is entered upon completion of a dma operation.
  * At this point the current dma_buffer is full and we can write it to
  * disk.  We set the "available data" pointer to point to the just-
  * filled buffer, and start the next dma operation on the other
  * buffer.  At the conclusion of operations, we output a non-specific
  * end-of-interrupt to the interupt controller.
  *
  * The PC bus provides no mechanism for "unlaching" an interrupt request
  * once it has been serviced.  In order to enable the next interrupt,
  * the hardware must be designed so that the request can be reset.  For
  * example, a write to an i/o port.  The external routine reset_irq()
  * must be coded to perform this routine.
  *
  * Declaring this routine as type 'interupt', ensures that all registers
  * are saved, the C data segment is set correctly, and that the routine
  * returns with an IRET instruction.  Further interrupts are disabled
  * during the execution of this routine.
  */

 curr_buf = dma_buffers[buf_index-1].p; /* Post just-filled buffer addreses */
 buf_index += 1;                       /* Increment buffer index */
 if (buf_index > numbuffers)
 {
  bufs_filled = 1;
  irq_flag = 2;
  init_brd();                          /* Initialize A/D board */
  outp(INTA00, EOI);                         /* Signal end of interupt */
  return;
  }
  dma_setup();                              /* Set up next dma operation */
  start_dma(dma_buffers[buf_index].p,dma_buffers[buf_index].s);
                                            /* Start dma on next buffer */
  irq_flag = 1;
  dma_chan ^= 2;                            /* Toggle DMA channel */
  inp(0x302);                   /* Reset brd. - sets U1A (-Y2) on A/D board */
  outp(INTA00, EOI);                        /* Signal end of interupt */
}
/*----------------------------------------------------------------------*/
void start_dma(buf,count)          /* Start a dma operation */
char far *buf;                     /* Address of buffer to be filled */
unsigned count;                         /* Size of buffer in bytes */
{
 int page;
 unsigned long addr =                   /* 20-bit address of dma buffer */
   FP_OFF(buf) + ((long) FP_SEG(buf) << 4);

 /* This routine starts a dma operation.  It needs to know :
  *   - the address where the dma buffer starts;
  *   - the number of bytes to tranfer
  * The dma buffer address is supplied in segmented, far-pointer
  * form (as returned by alloc_dma_buf()).  In this routine it is
  * converted to a 20-bit address by combining the segment and offset.
  * The upper four bits are known as the page number, and are handled
  * separately from the lower 16 bits.  The transfer count is
  * decremented by 1 because the dma controller reaches terminal count
  * when the count rolls over from 0000 to ffff.
  *
  * The dma transfer stops when the channel reaches terminal count.
  * The terminal count signal is turned around in the interface
  * hardware to reproduce an interrupt when dma is complete.
  *
  * Channels 4-7 are on a separate dma controller, available on
  * the PC-AT only.  They perform 16-bit transfers instead of 8-bit
  * transfers, and they are addressed in words instead of bytes.
  * This routine handles the addressing requirements based on the
  * channel number.
  *
  * dma_setup() needs to be called before start_dma() in order to
  * assign values to maskreg, modereg, etc.
  */

 page = PAGE(addr);            /* Extract upper bits of address */
 if(dma_chan >= 4)             /* For word-orientated channels: */
```

```
  {
   count >>= 1;              /* convert count to words */
   addr >>= 1;               /* convert address to words */
   page &= 0x7e;             /* address bit 16 is now in 'addr' */
  }
  count--;                   /* Compute count-1 (xfr stops at ffff) */
  outp(maskreg, sel | 0x04); /* Set mask bit to disable dma */
  outp(modereg, sel | 0x44); /* xfr mode (sngl, inc, noinit, write) */
  outp(basereg, LSB(addr));  /* O/p base address LSB */
  outp(basereg, MSB(addr));  /* O/p base address MSB */
  outp(pagereg, page);       /* O/p page number to page register */
  outp(cntreg,  LSB(count)); /* O/p count LSB */
  outp(cntreg,  MSB(count)); /* O/p count MSB */
  inp(statreg);              /* Clear DMA T/C information register */
  outp(maskreg, sel);        /* Clear mask bit, enabling dma */
 }
/*--------------------------------------------------------------------*/
void init_brd()          /* Initialize A/D board */
{
 inp(0x300); /* Address to set U1A (-Y0) on A/D board */
 delay(50); /* Wait for 50 msec */
 inp(0x300); /* Address to set U1A (-Y0) on A/D board */
 delay(50); /* Wait for 50 msec */
}
/*--------------------------------------------------------------------*/
void on_brd()           /* Turn A/D board on */
{
 inp(0x301); /* Turn board on */
}
```

"Plot.c" Source Program

```
/*
------------------------------------------------------------------
                 GENERIC X-Y PLOTTING ROUTINES

Plot description:  Plots data on user's screen.  Requires running "st_up_plt"
to generate graphics window, etc.  Can use this routine to continue
plotting previous plots.

Program inputs to plot:
     float x[10] y[10]   1-D arrays for x & y data
     int numpts;        Number of data points to plot
     int last plot     Graphics flag used to shut down graphics
                        after when finished (0 - leave graphics
                        on, 1 - shut graphics off)

Set-up_plot description:  Creates 1, 2, or 4 plots simutaneously on the user's
screen.  Requires a file that contains axis labeling
information.  File format:

     1. Graph title
     2. y-axis label
     3. x-axis label
     5. Number of x-axis increments
     6. x-axis scaling factor
     7. Number of y-axis increments
     8. y-axis scaling factor

Program inputs tp set_up_plot:
     int max_x,min_x,max_y,min_y
     int numpts:      Number of data points
     char file1[25]:   Data file name
     char file2[25]:   Axis labeling file name
     char pltpos[3]:   Plot position/size: "ul", "ur", "ll" ,
                      & "lr" are for 1/4 size and "upper",
                      "lower", "right", or "left" positions.
                      "u" or "l" are for 1/2 size and "upper" or "lower"
                      positions.
                      "c" is for full size and center position.
     int del_old_plt:  Graphics flag used to keep current plots
                      (0 - erase old plots, 1 - keep old plots)

Programmer:  Patrick T. Marshall
```

```
  Organization:  WRDC/AAWP-2
                 WPAFB, OH 45433-6543
  Date:  15 Nov 90
  Language:  Turbo C ver. 2.0

*/

#include <c:\borlandc\thesis\plot.h>

unsigned linestyle = 4;
struct linesettingstype oldlsetting;
unsigned color[16] = {2,3,4,5,6,7,8,9,10,11,12,13,14,15};
FILE   *fptr;                              /* File pointer */


                /********* Programs begin ***********/
label_plot(label_struct labels[16],int numlabels,char pltpos[])
{
 int i,max_length=0,length=0,oldcolor,x_offset=0,y_offset=0;
 int height=0;
 struct viewporttype cur_view;

 oldcolor = getcolor();
 getviewsettings(&cur_view);
 getlinesettings(&oldlsetting);
 settextstyle(SMALL_FONT,HORIZ_DIR,5);
 for(i=0;i<numlabels;i++)
 {
  length = textwidth(labels[i].l);
  height = textheight(labels[i].l);
  /* Add on for line key */
  max_length = max(max_length,length);
 }
 x_offset = cur_view.right-cur_view.left-max_length;
 y_offset = 5;
 for(i=0;i<numlabels;i++)
 {
  outtextxy(x_offset,y_offset+i*height,labels[i].l);
  /* Draw line key */
  if ((labels[i].t == "l") || (labels[i].t == "L"))
  {
   setcolor(labels[i].lt);
   setlinestyle(USERBIT_LINE,linestyle,NORM_WIDTH);
   line(.75*maxwidth+max_length+3-x_offset,13*i+10,.75*maxwidth+
                                          max_length+9,13*i+10);
   }
  }
  setlinestyle(oldlsetting.linestyle,oldlsetting.upattern,oldlsetting.thickness);
  setcolor(oldcolor);
  return;
}


print_plot()
{
  int errorcode,h,v;

  maxheight = getmaxy();
  maxwidth = getmaxx();
  setviewport(0,0,maxwidth,maxheight,1);
  InitGraf(&h,&v);
  printimage(0,0,h,v);
  errorcode = graphresult();
  if (errorcode != grOk)        /* Checking for graphics error */
  {
    printf("Graphics error: %s\n",grapherrormsg(errorcode));
    printf("Location: print_plot() routine in plot3.c\n");
    exit(1);
  }
  return 0;
}

plot(float huge *y,float x_inc,float x_beg,unsigned long numpts,
         int line_type)
{
  unsigned long i,errorcode;
  int X1,Y1,X2,Y2,oldcolor;

  if (line_type > 15) line_type = 15;
  if (line_type < 0) line_type = 0;
  oldcolor = getcolor();
  Y1 = m_y * y[0] + b_y;
  X1 = m_x * x_beg + b_x;
  getlinesettings(&oldlsetting);
  setcolor(color[line_type]);
  for (i=1;i<numpts;i++)
  {
    Y2 = m_y * y[i] + b_y;
    X2 = m_x * (i*x_inc+x_beg) + b_x;
    line(X1,Y1,X2,Y2);
    X1 = X2;Y1 = Y2;
   }
   errorcode = graphresult();
   if (errorcode != grOk)        /* Checking for graphics error */
   {
    closegraph();
    printf("Graphics error: %s\n",grapherrormsg(errorcode));
    printf("Location: plot() routine in plot3.c\n");
    exit(1);
   }
   setlinestyle(oldlsetting.linestyle,oldlsetting.upattern,oldlsetting.thickness);
   setcolor(oldcolor);
   return;
  }

 plot2(float x[],float y[],unsigned numpts,int line_color)
 {
  unsigned long int i,errorcode;
  int X1,Y1,X2,Y2,oldcolor;

  if (line_color > 13) line_color = 13;
  if (line_color < 0) line_color = 0;
  oldcolor = getcolor();
  getlinesettings(&oldlsetting);
  setcolor(color[line_color]);
  for (i=0;i<numpts-1;i++)
  {
   Y1 = m_y * y[i] + b_y;
   X1 = m_x * x[i] + b_x;
   Y2 = m_y * y[i+1] + b_y;
   X2 = m_x * x[i+1] + b_x;
   line(X1,Y1,X2,Y2);
  }
 errorcode = graphresult();
 if (errorcode != grOk)        /* Checking for graphics error */
 {
  closegraph();
  printf("Graphics error: %s\n",grapherrormsg(errorcode));
  printf("Location: plot2() routine in plot3.c\n");
  exit(1);
 }
 setlinestyle(oldlsetting.linestyle,oldlsetting.upattern,oldlsetting.thickness);
 setcolor(oldcolor);
 return;
}
```

```
set_up_plt(file,max_x,min_x,max_y,min_y,pltpos)
float max_x,min_x,max_y,min_y;
char file[],pltpos[];
{
 void find_precision(float,int*);
 float rnd(float,logical,int);
 double fraction,integer;
 float x_value,y_value,prev_y_value;
 float y_scale,x_scale,x_step_value,y_step_value;
 int errorcode;          /* Graphics error code */
 int x,y,prev_y,x_steps,y_steps,step_x,step_y,baseline;
 int plot_height,plot_width;
 int X1,X2,Y1,Y2,precision;
 int y_fudge,x_fudge,y_border=20,x_border=60,max_len:
 int significant_digits;
 char buffer[81];
 int length,n;
 size_t len;
 logical sign_change = FALSE;
 struct viewporttype cur_view;

 setbkcolor(1);
 maxheight = getmaxy();
 maxwidth = getmaxx();
 if ((fptr = fopen(file,"r"))  == NULL)
 {
  perror("fopen #1 failed\n");
  printf("Location: set_up_plt() routine in plot3.c\n");
  exit(1);
 }
 if (stricmp(pltpos,"ul") == 0)
 {
  plot_width = maxwidth/2;
  plot_height = maxheight/2;
  Y1 = 0;
  X1 = 0;
  Y2 = maxheight/2;
  X2 = maxwidth/2;
  significant_digits = 3;
 }
 if (stricmp(pltpos,"ur") == 0)
 {
  plot_width = maxwidth/2-2;
  plot_height = maxheight/2;
  Y1 = 0;
  X1 = maxwidth/2;
  Y2 = maxheight/2;
  X2 = maxwidth-2;
  significant_digits = 3;
 }
 if (stricmp(pltpos,"ll") == 0)
 {
  plot_width = maxwidth/2;
  plot_height = maxheight/2;
  Y1 = maxheight/2;
  X1 = 0;
  Y2 = maxheight;
  X2 = maxwidth/2;
  significant_digits = 3;
 }
 if (stricmp(pltpos,"lr") == 0)
 {
  plot_width = maxwidth/2-2;
  plot_height = maxheight/2;
  Y1 = maxheight/2;
  X1 = maxwidth/2;
  Y2 = maxheight;
  X2 = maxwidth-2;
  significant_digits = 3;
 }
 if (stricmp(pltpos,"u") == 0)
 {
  plot_width = maxwidth-2.5*x_border;
  plot_height = maxheight/2;
  Y1 = 0;
  X1 = x_border;
  Y2 = maxheight/2;
  X2 = maxwidth-1.5*x_border;
  significant_digits = 5;
 }
 if (stricmp(pltpos,"l") == 0)
 {
  plot_width = maxwidth-2.5*x_border;
  plot_height = maxheight/2;
  Y1 = maxheight/2;
  X1 = x_border;
  Y2 = maxheight;
  X2 = maxwidth-x_border;
  significant_digits = 5;
 }
 if (stricmp(pltpos,"c") == 0)
 {
  plot_width = maxwidth-2;
  plot_height = maxheight;
  Y1 = 0;
  X1 = 0;
  Y2 = maxheight;
  X2 = maxwidth-2;
  significant_digits = 5;
 }
 setviewport(X1,Y1,X2,Y2,1);
 rectangle(0,0,plot_width,plot_height);
                         /* Graph title */
 settextstyle(SMALL_FONT,HORIZ_DIR,5);
 fgets(buffer,80,fptr);fgets(buffer,80,fptr);
 length = textwidth(buffer);
 outtextxy(plot_width/2-length/2,5,buffer);
                         /* Y-axis label */
 settextstyle(SMALL_FONT,VERT_DIR,4);
 fgets(buffer,80,fptr);fgets(buffer,80,fptr);
 length = textwidth(buffer);
 outtextxy(0,plot_height/2-length/2,buffer);
                         /* X-axis label */
 settextstyle(SMALL_FONT,HORIZ_DIR,4);
 fgets(buffer,80,fptr);fgets(buffer,80,fptr);
 length = textwidth(buffer);
 outtextxy(plot_width/2-length/2,plot_height-15,buffer);
 fgets(buffer,80,fptr);fscanf(fptr, "%d\n",&x_steps);
 fgets(buffer,80,fptr);fscanf(fptr, "%f\n",&x_scale);
 fraction = modf((double)min_x,&integer);
 if(fabs(integer) > 1) min_x = integer;
 x_step_value = ((double)((max_x-min_x)/((double) x_steps)*x_scale));
 find_precision(fabs(x_step_value),&precision);
 if(fabs(min_x) != max_x) x_step_value = rnd(x_step_value,UP,precision);
 x_value = rnd(min_x*x_scale,DOWN,precision);
 step_x = (plot_width-1.5*x_border)/((double) x_steps);
 x_fudge = plot_width-1.5*x_border-x_steps*step_x;
 x = x_border;
 fgets(buffer,80,fptr);fscanf(fptr, "%d\n",&y_steps);
 fgets(buffer,80,fptr);fscanf(fptr, "%f\n",&y_scale);
 y_step_value = ((double)((max_y-min_y)/y_steps*y_scale));
 find_precision(fabs(y_step_value),&precision);
 if(fabs(min_y) != max_y) y_step_value = rnd(y_step_value,UP,precision);
```

```
y_value = rnd(max_y*y_scale,UP,precision);
step_y = (plot_height-2*y_border)/((double) y_steps);
y_fudge = plot_height-2*y_border-y_steps*step_y;
y = y_border;
X2 = X1;                    /* Modifying for center rectangle */
X1 += x;
for (n = 0;n < x_steps+1;n++) /* Creating and labeling x axis tick marks */
{
 if(fabs(x_value) < 1.0e-04) x_value = 0;
 line(x,plot_height-y_border-2-y_fudge,x,plot_height-y_border+2-y_fudge);
 gcvt(x_value,significant_digits,buffer);
 len = strlen(buffer);
 if(len > 1)
 {
  if (buffer[0] == '-') len = 3*len;
  else len = 2*len;
 }
 outtextxy(x-len,plot_height-y_border-y_fudge+5,buffer);
 x += step_x;
 x_value = x_value + x_step_value;
}
x -= step_x;
X2 += x;                    /* For center rectangle */
plot_width= X2 - X1;
max_x = x_value - x_step_value;
significant_digits = 4;
max_len = 0;
for (n = 0;n < y_steps+1;n++) /* Find average y buf size in chars */
{
 if(fabs(y_value) < 1.0e-04) y_value = 0;
 gcvt(y_value,significant_digits,buffer);
 len = strlen(buffer);
 max_len = max(max_len,len);
 y = y + step_y;
 y_value = y_value - y_step_value;
}
y = plot_height - y_border - y_fudge;
prev_y = y;
y_value = rnd(min_y*y_scale,DOWN,precision);
prev_y_value = y_value;
Y2 = Y1;                    /* Modifying for center rectangle */
Y2 += y;
for (n = y_steps+1;n > 0;n--) /* Creating and labeling y axis tick marks */
{
 if(fabs(y_value) < 1.0e-04) y_value = 0;
```

```
    line(x_border-2,y,x_border+2,y);
    gcvt(y_value,significant_digits,buffer);
    len = strlen(buffer);
    outtextxy(x_border-20-2*(len+max_len),y-5,buffer);
    y = y - step_y;
    y_value = y_value + y_step_value;
    if ((y_value < 0.0) && ((y_value+step_y) >= 0.0))
    {
     baseline = -prev_y_value*(y-prev_y)/(y_value-prev_y_value)+prev_y;
     sign_change = TRUE;
    }
    prev_y_value = y_value;
    prev_y = y;
   }
   y += step_y;
   Y1 += y;                    /* For center rectangle */
   plot_height = Y2 - Y1;
   max_y = y_value - y_step_value;
   baseline = baseline - y_border;
   setviewport(X1,Y1,X2,Y2,1);
   clearviewport();
   rectangle(0,0,plot_width,plot_height);
   setlinestyle(DASHED_LINE,0,NORM_WIDTH);
   if (sign_change == TRUE) line(0,baseline,plot_width,baseline);
   setlinestyle(SOLID_LINE,0,NORM_WIDTH);
   settextstyle(SMALL_FONT,HORIZ_DIR,3);
                          /* Graphical slope and offset conversion factors */
   getviewsettings(&cur_view);
   errorcode = graphresult();
   if (errorcode != grOk)      /* Checking for graphics error */
   {
    closegraph();
    printf("Graphics error: %s\n",grapherrormsg(errorcode));
    printf("Location: set_up_plt() routine in plot3.c\n");
    exit(1);
   }
   m_x = (cur_view.right-cur_view.left)/(max_x-min_x);
   b_x = (cur_view.right-cur_view.left) - m_x*max_x;
   m_y = -(cur_view.bottom-cur_view.top)/(max_y-min_y);
   b_y = 0 - m_y * max_y;
   fclose(fptr); /* The following are used for "switch_windows()" */
   if (stricmp(pltpos,"c") == 0)
   {
    c[0] = X1;c[1] = Y1;c[2] = X2;c[3] = Y2;
    cm_x = m_x;cm_y = m_y,cb_x = b_x;cb_y = b_y;
```

```
 if (sign_change == TRUE) c_baseline = baseline;
 else c_baseline = -1;
}
if (stricmp(pltpos,"u") == 0)
{
 u[0] = X1;u[1] = Y1;u[2] = X2;u[3] = Y2;
 um_x = m_x;um_y = m_y,ub_x = b_x;ub_y = b_y;
 if (sign_change == TRUE) u_baseline = baseline;
 else u_baseline = -1;
}
if (stricmp(pltpos,"l") == 0)
{
 l[0] = X1;l[1] = Y1;l[2] = X2;l[3] = Y2;
 lm_x = m_x;lm_y = m_y,lb_x = b_x;lb_y = b_y;
 if (sign_change == TRUE) l_baseline = baseline;
 else l_baseline = -1;
}
if (stricmp(pltpos,"ur") == 0)
{
 ur[0] = X1;ur[1] = Y1;ur[2] = X2;ur[3] = Y2;
 urm_x = m_x;urm_y = m_y,urb_x = b_x;urb_y = b_y;
 if (sign_change == TRUE) ur_baseline = baseline;
 else ur_baseline = -1;
}
if (stricmp(pltpos,"ul") == 0)
{
 ul[0] = X1;ul[1] = Y1;ul[2] = X2;ul[3] = Y2;
 ulm_x = m_x;ulm_y = m_y,ulb_x = b_x;ulb_y = b_y;
 if (sign_change == TRUE) ul_baseline = baseline;
 else ul_baseline = -1;
}
if (stricmp(pltpos,"ll") == 0)
{
 ll[0] = X1;ll[1] = Y1;ll[2] = X2;ll[3] = Y2;
 llm_x = m_x;llm_y = m_y,llb_x = b_x;llb_y = b_y;
 if (sign_change == TRUE) ll_baseline = baseline;
 else ll_baseline = -1;
}
if (stricmp(pltpos,"lr") == 0)
{
 lr[0] = X1;lr[1] = Y1;lr[2] = X2;lr[3] = Y2;
 lrm_x = m_x;lrm_y = m_y,lrb_x = b_x;lrb_y = b_y;
 if (sign_change == TRUE) lr_baseline = baseline;
 else lr_baseline = -1;
}
```

```
 return;
}

void switch_windows(char pltpos[])
{
 if (stricmp(pltpos,"c") == 0)
 {
  setviewport(c[0],c[1],c[2],c[3],1);
  m_x = cm_x;m_y = cm_y,b_x = cb_x;b_y = cb_y;
 }
 if (stricmp(pltpos,"u") == 0)
 {
  setviewport(u[0],u[1],u[2],u[3],1);
  m_x = um_x;m_y = um_y,b_x = ub_x;b_y = ub_y;
 }
 if (stricmp(pltpos,"l") == 0)
 {
  setviewport(l[0],l[1],l[2],l[3],1);
  m_x = lm_x;m_y = lm_y,b_x = lb_x;b_y = lb_y;
 }
 if (stricmp(pltpos,"ur") == 0)
 {
  setviewport(ur[0],ur[1],ur[2],ur[3],1);
  m_x = urm_x;m_y = urm_y,b_x = urb_x;b_y = urb_y;
 }
 if (stricmp(pltpos,"ul") == 0)
 {
  setviewport(ul[0],ul[1],ul[2],ul[3],1);
  m_x = ulm_x;m_y = ulm_y,b_x = ulb_x;b_y = ulb_y;
 }
 if (stricmp(pltpos,"ll") == 0)
 {
  setviewport(ll[0],ll[1],ll[2],ll[3],1);
  m_x = llm_x;m_y = llm_y,b_x = llb_x;b_y = llb_y;
 }
 if (stricmp(pltpos,"lr") == 0)
 {
  setviewport(lr[0],lr[1],lr[2],lr[3],1);
  m_x = lrm_x;m_y = lrm_y,b_x = lrb_x;b_y = lrb_y;
 }
}

void clear_plots(char pltpos[])
{
 struct viewporttype cur_view;
 int plot_height,plot_width,oldcolor;

 getviewsettings(&cur_view);
 clearviewport(); /* Now have to redraw border & dashed line (if any) */
 plot_height = cur_view.bottom-cur_view.top;
 plot_width = cur_view.right-cur_view.left;
 rectangle(0,0,plot_width,plot_height);
 setlinestyle(DASHED_LINE,0,NORM_WIDTH);
 if((stricmp(pltpos,"c") == 0) && (c_baseline != -1))
        line(0,c_baseline,plot_width,c_baseline);
 if((stricmp(pltpos,"u") == 0) && (u_baseline != -1))
        line(0,u_baseline,plot_width,u_baseline);
 if((stricmp(pltpos,"l") == 0) && (l_baseline != -1))
        line(0,l_baseline,plot_width,l_baseline);
 if((stricmp(pltpos,"ur") == 0) && (ur_baseline != -1))
        line(0,ur_baseline,plot_width,ur_baseline);
 if((stricmp(pltpos,"ul") == 0) && (ul_baseline != -1))
        line(0,ul_baseline,plot_width,ul_baseline);
 if((stricmp(pltpos,"ll") == 0) && (ll_baseline != -1))
        line(0,ll_baseline,plot_width,ll_baseline);
 if((stricmp(pltpos,"lr") == 0) && (lr_baseline != -1))
        line(0,lr_baseline,plot_width,lr_baseline);
 setlinestyle(SOLID_LINE,0,NORM_WIDTH);
}

erase_plot()
{
 closegraph();
 return;
}

init_plot()
{
 int driver=DETECT,mode; /* Graphics driver and mode */
 int errorcode;

                /* Determine and setup graphics hardware */
 detectgraph(&driver,&mode);
 if (driver < 0)
 {
  printf("No graphics hardware available!\n");
  printf("Graphics driver error code = %d\n",driver);
  printf("Location: init_plot() routine in plot3.c\n");
  exit(1);
 }
 errorcode=graphresult();
 if (errorcode != grOk)        /* Checking for graphics error */
 {
  printf("Graphics error: %s\n",grapherrormsg(errorcode));
  printf("Location: init_plot() routine in plot3.c\n");
  exit(1);
 }
                /* Now create the x-y graph */
 initgraph(&driver, &mode, "c:\\borlandc\\bgi");
 if (driver < 0)
 {
   perror("Graphics Error!\n");
   printf("Graphics Driver = %d\n",driver);
   printf("Location: init_plot() routine in plot3.c\n");
   exit(1);
 }
 return;
}


/*********************************************************************
 *********************************************************************
  Round input value

      Inputs:
             V - value to be rounded
      Outputs:
             V - rounded value
*********************************************************************/
float rnd(float V,logical direction,int precision)
{
 double fraction,integer,temp1,temp2;
 int count;

 fraction = modf((double)V,&integer);
 temp1 = modf(fraction*pow(10,precision),&temp2);
 if(fabs(temp1) < 1.0e-04) temp1 = 0;
 if(direction == UP) temp1 = ceil(temp1);
 else temp1 = floor(temp1);
 V = integer + (temp1+temp2)/pow(10,precision);
 return(V);
}

void find_precision(float del,int *precision)
{
```

```
int count;

if(del>=1)
{
 count = 3;
 do
 {
  del /= 10.0;
  count --;
  if(del < 1) break;
 }
 while(count >= 0);
}
else
{
 count = 0;
 do
 {
  del *= 10.0;
  count ++;
 }
 while(del < 1);
}
precision[0] = count;
return;
}
```

"Proc_spc.c" Source Program

```
/**************************************************************************
 Data processing program that strips low freq photo data from AM data.
 Also, microphone data is converted to FFT data

 Program: proc_spc.c
 Programmer: Patrick T. Marshall
 Date: 11/19/91
 Organization: WRDC/AAWP-2,
               WPAFB, OH 45433
 Phone: (513) 255-2471

**************************************************************************/
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\plot.h"
#include "c:\borlandc\thesis\proc_spc.h"

void set_up_plt(char[],double,double,double,double,char[]);
void plot(float huge *,float,float,unsigned long,int);
void plot2(float huge*,float huge*,unsigned,int);
void label_plot(label_struct[],int,char[]);
void print_plot(void);
void erase_plot(void);
void init_plot();
extern int  beg_mult,end_mult;


/**************************************************************************
 The following processes photo words found in recorded speech and returns
```

```
results. This routine uses the "find_data()" routine to strip data from
the recorded speech data file. It then decreases the number of data
points by skipping data. Decreasing data is required due to a memory
limitation while using the DTW algorithm.

Inputs:
  filename - complete file name (i.e. contains complete path & drive).
  beg_time - beginning time of word
  end_time - ending time of word
  mult_num - word multiple number

Outputs:
  n - size of output buffer
  out_buf - processed output buffer

Global:
  complete_file_name - filename to be processed (includes path & drive)
  start_time & stop_time - used in convert_data()

*************************************************************************/
float *process_pho_data(char filename[],float beg_time,float end_time,
                        unsigned mult_num,unsigned *n)
{
 float *find_data(char[],char[],float,float,unsigned,unsigned*,
        unsigned long*);
 float *filter_data(float huge*,unsigned long,unsigned long*);
 float *out_buf;
 unsigned long count=0,beg_n=0,end_n=0;
 unsigned max_mult_num,skip_intv=0;

 strcpy(complete_file_name,filename);
 get_file_info(&max_mult_num,&numbytes);
 /* where: max_mult_num = number of words and/or sentences &  */
 /*        numbytes = window size used to record data */
 close(file_handle); /* "file_handle" is a global variable used by */
                     /* get_file_info to open files and leave open */
 start_time = 0;stop_time = numbytes/22.0/2500.0;
 beg_n=(unsigned long)(beg_time*2500);end_n=(unsigned long)(end_time*2500);
 skip_intv = (unsigned)(end_n-beg_n)/100;
 out_buf = find_data(filename,"p",beg_time,end_time,mult_num,&skip_intv,
           &count);
 out_buf = filter_data(out_buf,count,&count);      /* Smooth out peaks */
 n[0] = (unsigned)count;
 return((void *)out_buf);
}
```

```
/*************************************************************************
 The following processes mic. words found in speech and returns results.
 This routine uses the "find_data()" routine to strip data from the recorded
 speech data file. The data is then converted to FFT data which is used to
 find max peaks w/i certain filter banks. Lastly, this algorithm will then
 decrease the number of data points by skipping data. Decreasing data is
 required due to a memory limitation while using the DTW algorithm.

 Inputs:
  filename - complete file name (i.e. contains complete path & drive).
  beg_time - beginning time of word
  end_time - ending time of word
  mult_num - word multiple number

 Outputs:
  n - size of output buffer
  out_buf - processed output buffer

 Global:
  complete_file_name - filename to be processed (includes path & drive)
  start_time & stop_time - used in convert_data()
*************************************************************************/
float *process_mic_data(char filename[],float beg_time,float end_time,
                        unsigned mult_num,unsigned *n)
{
 float *find_words(char[],unsigned*,char,unsigned);
 float *find_data(char[],char[],float,float,unsigned,unsigned*,
                  unsigned long*);

 unsigned long i=0,ii=0,beg_i=0,end_i=0,max_par,count=0;
 unsigned long max_ints,M,num_ints;
 unsigned long INV,mem_avail,prev_buf_size;
 unsigned long buf_size,skip_flag,max_mult_num;
 unsigned finish_flag,numwords,skip_int;
 float huge *out_buf,max_amp,pitch_freq;
 float fs,freq_res,huge *real_arr,huge *img_arr;
 float volts[1],amplitude,freq,prev_amp,freq_count;
 float ave_amplitude,filter_BW,huge *in_buf;
 FILE *tmpfile;
 char pltpos[3],plot_info[50],p_buffer[12];
 label_struct labels[16];

 strcpy(complete_file_name,filename);
```

```
get_file_info(&max_mult_num,&numbytes);
/* where: max_mult_num = number of words and/or sentences &  */
/*        numbytes = window size used to record data */
close(file_handle); /* "file_handle" is a global variable used by */
                    /* get_file_info to open files and leave open */
start_time = 0;stop_time = numbytes/22.0/2500.0;
beg_i=(unsigned long)(beg_time*25000.0);
end_i=(unsigned long)(end_time*25000.0);
fs = 25000.0;                 /* Mic sampling freq */
num_ints = end_i-beg_i;
max_ints = (unsigned long)farcoreleft()/4/4;
M = (unsigned long)floor(log10(max_ints)/log10(2));
max_ints = (unsigned long)pow(2,M);
skip_int = (unsigned)ceil((float)num_ints/(float)max_ints);
if ((tmpfile = fopen("c:\\borlandc\\thesis\\tmp.dat","wb+")) == NULL)
{
 printf("ERROR(1):  data file open failed!\n");
 printf("Location:  iso_mic_fft routine in proc_spc.c.\n");
 perror(" ");
 exit(1);
}
out_buf = find_data(filename,"m",beg_time,end_time,mult_num,&skip_int,
           &count);
fs=(float)count/(end_time-beg_time);  /* Calc new fs based on skipped data */
fwrite((void *)out_buf,sizeof(float),count,tmpfile);
farfree((void*)out_buf);
rewind(tmpfile);
if ((real_arr = farcalloc(max_ints+1,sizeof(float))) == NULL)
{
 printf("ERROR(1):  real array memory allocation failed.\n");
 printf("num_ints = %lu \n",num_ints);
 perror("");
 printf("Location: proc_spc() routine in proc_spc.c.\n");
 farfree((void *)real_arr);
 exit(1);
}
fread((void *)real_arr,sizeof(float),count,tmpfile);
fclose(tmpfile);
if ((img_arr = farcalloc(max_ints+1,sizeof(float))) == NULL)
{
 printf("ERROR(2):  img array memory allocation failed.\n");
 printf("num_ints = %lu \n",num_ints);
 perror("");
 printf("Location: proc_spc() routine in proc_spc.c.\n");
 farfree((void *)real_arr);
```

```
  exit(1);
 }
 INV = FALSE;
 fft2(real_arr,img_arr,M,INV);
 for (i=0;i<max_ints;i++)
  real_arr[i] = sqrt(pow(real_arr[i],2)+pow(img_arr[i],2));
 farfree((void *)img_arr);
 freq_res = fs/(float)max_ints;max_amp=-5000;
 filter_BW=12;freq_count=0.0;freq=0.0;i=0;count=0;
 if ((out_buf = farcalloc(201,sizeof(float))) == NULL)
 {
  printf("ERROR(5):  out_buf buffer allocation failed!\n");
  printf("Location:  iso_mic_fft() routine in proc_spc.c.\n");
  perror("");
  exit(1);
 }
 ave_amplitude = 0.0;ii=0;count=0;freq_count = filter_BW;
 for(i=0;i<max_ints;i++)
 {
  freq = i*freq_res;
  if(freq>=2000) filter_BW = 60;
  if(freq<freq_count)
  {
   ave_amplitude += real_arr[i];
   count++;
  }
  else
  {
   ave_amplitude /= (float)count;
   out_buf[ii] = log10(ave_amplitude);
   ave_amplitude = 0.0;
   ii++;count = 0;
   freq_count += filter_BW;
  }
  if(ii>=201) break;
 }
 farfree((void *)real_arr);
 *n = 200;
 return((void *)out_buf);
}


/***************************************************************************
 The following is a low pass filter for smoothing data.  It performs
 a 5-point interpolation.
 Inputs:
        in_buf - data buffer to be filtered (smoothed)
        buf_size - size of in_buf
 Outputs:
        out_buf - filtered result of in_buf
        n - size of out_buf

*************************************************************************/
float *filter_data(float huge *in_buf,unsigned long buf_size,unsigned long *n)
{
 unsigned long i=0;
 float *out_buf,min_volts,max_volts;
 float threshold1,threshold2;
 char file[31]="",pltpos[3];

 strcpy(&file[0],"c:\\borlandc\\thesis\\irinc.dat");
 if ((out_buf = farcalloc(buf_size,sizeof(float))) == NULL)
 {
  printf("ERROR(1):  out_buf buffer allocation failed!\n");
  printf("Location:  filter_data() routine in proc_spc.c.\n");
  perror("");
  exit(0);
 }
 min_volts = 5.0;
 for (i=2;i<buf_size-2;i++)    /* Smooth out peaks */
 {
  out_buf[i] = (float)((in_buf[i-2] + 4.0*in_buf[i-1] + 6.0*in_buf[i] +
                       4.0*in_buf[i+1] + in_buf[i+2])/16.0);
  min_volts = min(out_buf[i-2],min_volts);
 }
 out_buf[1] = 1/4.0*in_buf[0]+1/2.0*in_buf[1]+1/4.0*in_buf[2];
 out_buf[buf_size-2] = 1/4.0*in_buf[buf_size-3]+1/2.0*in_buf[buf_size-2]+
                       1/4.0*in_buf[buf_size-1];
 out_buf[0] = 3/4.0*in_buf[0] + 1/4.0*in_buf[1];
 out_buf[buf_size-1] = 3/4.0*in_buf[buf_size-1] + 1/4.0*in_buf[buf_size-2];
 n[0] = buf_size;
 farfree((void *)in_buf);
 return((void *)out_buf);
}

/*************************************************************************
 The following is an energy normalization algorithm.  The input buffer
 (in_buf) is normalized w.r.t it's energy content.
 Inputs:
        n - number of data points in in_buf
        in_buf - buffer to be normalized
 Outputs:
        in_buf - normalized buffer
        maxy - max y value (normalized amplitude)
        maxn - n value at maxy

***************************************************************************/
float *normalize(unsigned n,float huge *in_buf,float *maxy,unsigned *maxn)
{
 unsigned i=0;
 float norm=0,min_y=1.0e+06;

                /* Find min. to set offset to zero */
 min_y=1.0e+06;
 for (i=0;i<n;i++) min_y = min(min_y,in_buf[i]);
 for (i=0;i<n;i++) in_buf[i] = in_buf[i] - min_y;
                /* Calc. denominator for enery normalization */
 for (i=0;i<n;i++)
   norm += pow(in_buf[i],2);
 norm = sqrt(norm);*maxy = -1e4;*maxn=0;
                /* Energy normalize data */
 for (i=0;i<n;i++)
 {
  in_buf[i] /= norm;
  maxy[0] = max(maxy[0],in_buf[i]);
  if(maxy[0] == in_buf[i]) maxn[0] = i;
 }
 return((void *)in_buf);
}

/***************************************************************************
 The following is called from find_words() routine.  Its purpose is to
 find the microphone energy function.

Input variables:
        data_buf - empty data buffer
        word_num - word multiple number (for convert_data() routine)

Output variables:
        data_buf - filled energy data buffer
        time_inv - time between intervals
        n - length of buffers
```

```
*************************************************************************/
float *energy(float *time_inv,unsigned long *n,unsigned mult_num)
{
 float huge *convert_data(unsigned *,unsigned,char[],unsigned long *);
 double fract,intpart,x;
 float huge *raw_dat,huge *data_buf,fs,window;
 float stop_t,energy,time;
 unsigned max_mult_num,finish_flag,data_buf_size,ii=0;
 unsigned long buf_size,count=0,i=0;
 char pltpos[2];

 get_file_info(&max_mult_num,&numbytes);
 /* where: max_mult_num = number of words and/or sentences &  */
 /*        numbytes = window size used to record data */
 close(file_handle); /* "file_handle" is a global variable used by */
                     /* get_file_info to open files and leave open */
 fs = 25000.0;window = 300.0;
 start_time = 0;stop_time = numbytes/22.0/2500.0;
 data_buf_size = ceil(stop_time*fs/window) + 1;
 if ((data_buf = farcalloc(data_buf_size,sizeof(float))) == NULL)
 {
  printf("ERROR(2):  data buffer allocation failed!\n");
  printf("Location:  find_mic_words() routine in cont_rec.c\n");
  perror("");
  exit(1);
 }
 stop_t = 0;count = 1;ii = 0;energy = 0.0;
 finish_flag = TRUE;  /* Tells convert_data 1st run */
 do  /* Average every window'th data point */
 {
  raw_dat = convert_data(&finish_flag,mult_num,"m",&buf_size);
  for(i=0;i<buf_size;i++)
  {
   energy += fabs(raw_dat[i]);
   ii++;
   x = (double)ii/window;
   fract = modf(x,&intpart);
   time += 1/fs;
   if(fract == 0)
   {
      energy /= window;
      data_buf[count] = energy;
      energy = 0.0;
      if(count == 1) time_inv[0] = time;
```

```
           count++;
           if(count >= data_buf_size)
           {
            finish_flag = TRUE;
            break;
           }
        }
        if(count >= data_buf_size)
        {
           finish_flag = TRUE;
           break;
        }
      }
      farfree((void *)raw_dat);
      if(count >= data_buf_size)
      {
       finish_flag = TRUE;
       break;
      }
    }
    while(finish_flag==FALSE);
    data_buf[0] = data_buf[1];
    for(i=1;i<count-1;i++) /* Interpolate between data to eliminate time phase */
    {  /* shift (i.e., eliminate time offset by shifting waveform to the right */
     data_buf[i] = (data_buf[i] + data_buf[i+1])/2.0; /* by time_inv[0]/2) */
    }
    *n = count;
    return((void *)data_buf);
   }


/*******************************************************************************
 ************ WORD BOUNDARY DETECTION SENSOR FUSION ROUTINE
*************

 The following is called from the "template.c" & "spch_rec.c" routines.
 Its purpose is to find the correct beginning & ending boundaries for a
 given word.  This routine compares both data types (mic time energy &
 photo time volts) to help decide where words begin & end.

 For isolated words this routine relies more heavily on the mic energy
 distribution to find words.  It searches the mic energy for the largest
 peak value & records its time.  It then checks other mic peaks to see
 if they are w/i +/- 0.2 sec of max peak.  If they are then this routine
 will treat the other peak(s) as part of the original max peak.  The
```

```
algorithm also searches the photo to find the corresponding max-peak value
that is w/i +/- 0.2 sec of max mic peak.  Lastly it adjusts photo
endpoints if necessary.

For continuous words this routine relies more heavily on the photo
amplitude to find words.  It matches peak photo values with peak
mic values.  For a each match this routine checks to see if the
photo beginning is lagging the mic beginning.  If it is by a certain
amount then it treats the mic word as two words.  Also if the endpoints
don't match w/i a certain amount the routine will again treat the mic
word as two words.

input variables:
 filename - complete file name (i.e. contains complete path & drive).
 op_mode - (i)solated or (c)ontinuous

Output variables:
 None

Output word time filenames:
 phowrdtm.0nn (nn = 00 to 16)
 micwrdtm.0nn (nn = 00 to 16)

Global:
 complete_file_name - filename to be processed (includes path & drive)
 start_time & stop_time - used in convert_data()
****************************************************************************/
void sensor_fusion_segmentor(char filename[],char op_mode)
{
 float *find_words(char[],unsigned*,char,unsigned);
 unsigned find_peak(float[],unsigned);
 void build_path (char*, char*, char*, int);

 float *pho_word_times,*mic_word_times,time,pho_peak_time,mic_peak_time;
 float min_time1,min_time2,mic_peak_value,del_beg,del_end,del_peak;
 float pho_beg_time,pho_end_time,mic_beg_time,mic_end_time;
 unsigned i,ii,iii,beg_i,end_i,num_pho_words,num_mic_words;
 unsigned mic_beg_num,mic_end_num,mic_peak_num;
 unsigned pho_beg_num,pho_end_num,mic_word,pho_word,max_mult_num;
 unsigned bad_photo_words = 0,extra_mic_words = 0,extra_pho_words = 0;
 unsigned pho_word_match;
 int peak_match_arr[30];
 extern char complete_file_name[];
 char file[50]="";
 FILE *phoptr,*micptr;
```

```
strcpy(complete_file_name,filename);            /* For get_file_info() */
get_file_info(&max_mult_num,&numbytes);
/* where: max_mult_num = number of words and/or sentences &  */
/*        numbytes = window size used to record data */
close(file_handle); /* "file_handle" is a global variable used by */
                    /* get_file_info to open files and leave open */
      /* The following is required in convert_data() routine */
start_time = 0;stop_time = numbytes/22.0/2500.0;
beg_i=beg_mult;end_i=end_mult;
for(i=0;i<30;i++) peak_match_arr[i] = -1;
for(i=beg_i;i<=end_i;i++)
{
 build_path(file,drive,"\\borlandc\\thesis\\phowrdtm",i);
 if((phoptr = fopen(file,"wb+")) == NULL)
 {                    /* Pointer to word endpoint time file */
  wait_message(0,7,"ERROR(1):  data file open failed!",
     "Location:  sensor_fusion_segmentor() routine in proc_spc.c");
  perror(" ");exit(0);
 }
 build_path(file,drive,"\\borlandc\\thesis\\micwrdtm",i);
 if((micptr = fopen(file,"wb+")) == NULL)
 {                    /* Pointer to word endpoint time file */
  wait_message(0,7,"ERROR(2):  data file open failed!",
     "Location:  sensor_fusion_segmentor() routine in proc_spc.c");
  perror(" ");exit(0);
 }
 pho_word_times = find_words(filename,&num_pho_words,'p',i);
 mic_word_times = find_words(filename,&num_mic_words,'m' i);
 min_time = 32000;bad_photo_words = 0;
/* Compare data type word times & find correct words beg.' & end.'s */
if(op_mode == 'c') /* Process continuous words - only work on mic. words */
{
 fwrite(&num_mic_words,sizeof(unsigned),1L,micptr);
    /* First get rid of mic words w/peaks amplitudes < 0.05 volts */
 mic_peak_value = mic_word_times[2];
 for(ii=0;ii<4*num_mic_words;ii+=4)
    mic_peak_value = max(mic_peak_value,mic_word_times[ii+2]);
 for(ii=0;ii<4*num_mic_words;ii+=4)
 {
    if(mic_word_times[ii+2] < 0.1*mic_peak_value)
    {      /* Found a bogus mic word */
     for(iii=ii;iii<4*num_mic_words;iii+=4)
     {                         /* Shift array values to the left */
      mic_word_times[iii] = mic_word_times[iii+4];    /* Beg. time */
```

```
          mic_word_times[iii+1] = mic_word_times[iii+5];  /* Peak time */
          mic_word_times[iii+2] = mic_word_times[iii+6];  /* Peak amp. */
          mic_word_times[iii+3] = mic_word_times[iii+7];  /* End. time */
        }
        num_mic_words--;ii-=4;     /* Have to compensate for shift */
      }
  }
/* Now find beg & end photo words compared to beg. & end mic words */
  for(ii=0;ii<4*num_pho_words;ii+=4)
  {
     time = fabs(mic_word_times[1] - pho_word_times[ii+1]);
     if(ii == 0) min_time1 = time;
     min_time1 = min(time,min_time1);
     if(min_time1 == time) pho_beg_num = ii;
     time = fabs(mic_word_times[4*(num_mic_words-1)+1]
                         - pho_word_times[ii+1]);
     if(ii == 0) min_time2 = time;
     min_time2 = min(time,min_time2);
     if(min_time2 == time) pho_end_num = ii;
  }
  for(ii=pho_beg_num;ii<=pho_end_num;ii+=4)
  {  /* Now write photo words for both mic & photo words */
     fwrite(&pho_word_times[ii],sizeof(float),1L,micptr);
     fwrite(&pho_word_times[ii+3],sizeof(float),1L,micptr);
  }
  rewind(micptr);   /* Overwrite mic file with new # of mic words */
  num_mic_words = (pho_end_num-pho_beg_num)/4+1;
  fwrite(&num_mic_words,sizeof(unsigned),1L,micptr);
 }
 else          /* Process isolated words */
 {
  for(ii=0;ii<4*num_mic_words;ii+=4)
  {  /* Match mic & photo words by comparing peak distances */
     mic_peak_time = mic_word_times[ii+1];
     min_time = 32000;
     for(iii=0;iii<4*num_pho_words;iii+=4)
     {
      pho_peak_time = pho_word_times[iii+1];
      time = fabs(pho_peak_time - mic_peak_time);
      min_time = min(time,min_time);
      if(min_time == time) pho_word_match = iii/4;
     }                              /* Matched pair */
     if(min_time < 0.2) peak_match_arr[ii/4] = pho_word_match;
  }  /* Find max mic peak */
  mic_peak_value = -5000.0;
```

```
for(ii=0;ii<4*num_mic_words;ii+=4)
{
   mic_peak_value = max(mic_peak_value,mic_word_times[4*ii+2]);
   if(mic_peak_value == mic_word_times[4*ii+2]) mic_peak_num = ii;
}
mic_beg_time = mic_word_times[4*mic_peak_num];
mic_end_time = mic_word_times[4*mic_peak_num+3];
pho_beg_time = pho_word_times[4*peak_match_arr[mic_peak_num]];
pho_end_time = pho_word_times[4*peak_match_arr[mic_peak_num]+3];
/* Check for multiple peaks & combine if found */
if((mic_peak_num > 0) && (num_mic_words > 1))
{                          /* Compare to previous peak */
   mic_peak_time = mic_word_times[4*(mic_peak_num-1)+1];
   if((mic_word_times[4*mic_peak_num+1] - mic_peak_time) <= 0.25)
   {  /* Change beg.time to previous peak's beg. time */
    mic_beg_time = mic_word_times[4*(mic_peak_num-1)];
    if(peak_match_arr[mic_peak_num-1] != -1) /* Adj. photo beg. also */
      pho_end_time = pho_word_times[4*peak_match_arr[mic_peak_num-1]];
   }
}
if(mic_peak_num < (num_mic_words - 1))
{                          /* Compare to next peak */
   mic_peak_time = mic_word_times[4*(mic_peak_num+1)+1];
   if((mic_word_times[4*mic_peak_num+1] - mic_peak_time) <= 0.25)
   {  /* Change beg.time to next peak's ending time */
    mic_end_time = mic_word_times[4*(mic_peak_num+1)+3];
    if(peak_match_arr[mic_peak_num+1] != -1) /* Adj. photo end also */
      pho_end_time = pho_word_times[4*peak_match_arr[mic_peak_num+1]+3];
   }
}
if((fabs(pho_beg_time - mic_beg_time)) > 0.1)
   pho_beg_time = mic_beg_time - 0.05;
if((fabs(pho_end_time - mic_end_time)) > 0.1)
   pho_end_time = mic_end_time + 0.05;
if(num_pho_words == 1)
{
   pho_beg_time = pho_word_times[0];
   pho_end_time = pho_word_times[3];
}
if(num_pho_words == 0)
{
   pho_beg_time = mic_beg_time - 0.05;
   pho_end_time = mic_end_time + 0.05;
}
if(pho_beg_time < 0.0) pho_beg_time = 0.0;
```

```
    if(pho_end_time > stop_time) pho_end_time = stop_time;
    num_pho_words = 1;num_mic_words = 1;
    fwrite(&num_pho_words,sizeof(unsigned),1L,phoptr);
    fwrite(&num_mic_words,sizeof(unsigned),1L,micptr);
    fwrite(&pho_beg_time,sizeof(float),1L,phoptr);
    fwrite(&pho_end_time,sizeof(float),1L,phoptr);
    fwrite(&mic_beg_time,sizeof(float),1L,micptr);
    fwrite(&mic_end_time,sizeof(float),1L,micptr);
   }
   free((void*)pho_word_times);free((void*)mic_word_times);
   fclose(phoptr);fclose(micptr);
  }
  return;
 }

/**************************************************************************
 The following is called from several routines. Its purpose is to find
 the peak word in a word array.

 Input variables:
   word_times - array contining word info. (beg. time, peak time, peak
                value, & end time)
   num_words - number of words in array

 Output variables:
   peak_word - word w/highest peak value
**************************************************************************/
unsigned find_peak(float word_times[],unsigned num_words)
{
 unsigned i,peak_word;
 float peak_value,max_value;

 max_value = -5000;
 for(i=0;i<num_words;i++)
 {
  peak_value = word_times[i*4+2];
  max_value = max(max_value,peak_value);
  if(max_value == peak_value) peak_word = i;
 }
 return(peak_word);
}

/**************************************************************************
 The following is called from several routines. Its purpose is to find
 the words in either photo volts or in mic. energy. Note that if this
 algorithm does not find a word(s) it returns "0" for number of words
 found and an empty "time_arr".

 Input variables:
  filename - complete file name (i.e. contains complete path & drive).
  type_data - (m)ic or (p)hoto data
  mult_num - word multiple number (for convert_data() routine)

 Output variables:
  time_arr -  Array w/beg. & end. times for beg. & ending points of words.
              The order is: (1) beginning time, (2) peak time, (3) peak
              amplitude, and (3) ending time.
  numwords - number of words found

 Global:
  complete_file_name - filename to be processed (includes path & drive)
  start_time & stop_time - used in convert_data()
************************************************************************/
float *find_words(char filename[],unsigned *numwords,char data_type,
                  unsigned mult_num)
{
 float *filter_data(float huge*,unsigned long,unsigned long*);
 float *energy(float*,unsigned long*,unsigned);
 float *normalize(unsigned,float huge *,float *,unsigned *);
 float huge *convert_data(unsigned *,unsigned,char[],unsigned long *);

 float huge *raw_dat,time,fs;
 float beg_time,end_time,del_ave=0,thershold,ave_back,ave_foward;
 float peak_time = 0,*time_arr,time_inv,max_value,min_value,min_y,max_y;
 unsigned long i,ii,iii,j,count,buf_size;
 unsigned max_mult_num,array_size,numread,win;
 int slope=0,prev_slope=0,store_slope[3];
 extern char complete_file_name[];
 char file[31]="";
 logical finish_flag = TRUE,beg_found = FALSE,peak_found=FALSE;
 logical end_found = FALSE;

 strcpy(complete_file_name,filename);              /* For get_file_info() */
 get_file_info(&max_mult_num,&numbytes);
 /* where: max_mult_num = number of words and/or sentences &  */
 /*        numbytes = window size used to record data */
 close(file_handle); /* "file_handle" is a global variable used by */
                     /* get_file_info to open files and leave open */
 if ((time_arr = farcalloc(180,sizeof(float))) == NULL)
 {
   printf("ERROR(1):  time_arr buffer allocation failed!\n");
   printf("Location:  find_pho_words() routine in cont_rec.c\n");
   perror("");
   exit(1);
  }
       /* The following is required in convert_data() routine */
  start_time = 0;stop_time = numbytes/22.0/2500.0;
  if(data_type == 'p')
  {
   fs = 2500.0;
   win = 20;                    /* Set window width for averaging data */
   thershold = 0.05;            /* Set slope voltage threshold */
  }
  else
  {
   win = 3;
   fs = 25000.0;
   thershold = 0.02;            /* Set slope voltage threshold */
  }
  time_inv = 1.0/fs;
  finish_flag = TRUE;
  min_y = 5000;max_y = -5000;
  do                            /* Find min & max values */
  {
   if(data_type == 'p')
    raw_dat = convert_data(&finish_flag,mult_num,"p",&buf_size);
   else
   {
    raw_dat = energy(&time_inv,&count,mult_num);
    raw_dat = filter_data(raw_dat,count,&count);   /* Smooth out peaks */
    raw_dat = filter_data(raw_dat,count,&count);
    buf_size = (long)count;
   }
   for(i=0;i<buf_size;i++)
   {
    min_y = min(min_y,raw_dat[i]);
    max_y = max(max_y,raw_dat[i]);
   }
   farfree((void *)raw_dat);
  }
 while(finish_flag==FALSE);
 for(i=0;i<3;i++) store_slope[i] = 0;max_value=-5000;min_value=5000;
 finish_flag = TRUE;ii = 0;iii=0;count = 0;
 time = (float)win*time_inv; /* Start time */
 beg_time=0.0;end_time=0.0;
```

```
do                                      /* Find beg. & end of words */
{
 if(data_type == 'p')
 {
  raw_dat = convert_data(&finish_flag,mult_num,"p",&buf_size);
  for(i=0;i<buf_size;i++) raw_dat[i] -= min_y;
 }
 else
 {
  raw_dat = energy(&time_inv,&count,mult_num);
  raw_dat = filter_data(raw_dat,count,&count);   /* Smooth out peaks */
  raw_dat = filter_data(raw_dat,count,&count);
  buf_size = (long)count;
 }
 for(i=win;i<buf_size-win;i++)              /* Find beg. & end. of words */
 {
  ave_back = 0.0;ave_foward = 0.0;
  for(ii=0;ii<win;ii++) ave_foward += raw_dat[i+ii];
  for(ii=0;ii<win;ii++) ave_back += raw_dat[i-ii];
  ave_foward /= (win-1);ave_back /= (win-1);
  del_ave = ave_foward - ave_back;
  prev_slope = slope;
  if(fabs(del_ave) < thershold) slope = 0;
  else if(del_ave >= thershold) slope = 1;
  if(del_ave <= -thershold) slope = -1;
  if(prev_slope != slope)                  /* Slope changed */
  {
     store_slope[0] = store_slope[1];
     store_slope[1] = store_slope[2];
     store_slope[2] = slope;
  }
  if((store_slope[1] <= 0) && (store_slope[2] == 1) &&
      (beg_found == FALSE))
  {                                        /* Found beginning */
     beg_found = TRUE;
     if(data_type == 'm') thershold = 0.01;
     if(peak_found == TRUE) peak_found = FALSE;
     else beg_time = time;
  }
  if((store_slope[0] == 0) && (store_slope[1] == 1) &&
      (store_slope[2] <= 0) &&
      (peak_found == FALSE) && (end_found != TRUE))
  {                                        /* Found peak */
     peak_found = TRUE;
     peak_time = time;
```

```
      max_value = raw_dat[i];
   }
   if((peak_found == TRUE) && (end_found == FALSE))
   {
      max_value = max(max_value,raw_dat[i]);
      if(max_value == raw_dat[i]) peak_time = time; /* Found a better peak */
   }
   if((store_slope[0] >= 0) && (store_slope[1] == -1) &&
      (store_slope[2] >= 0) && (peak_found == TRUE))
   {                                             /* Found end */
      beg_found = FALSE;
      peak_found = TRUE;
      end_found = TRUE;
      end_time = time;
      for(j=0;j<3;j++) store_slope[j] = 0;
      prev_slope = 0;slope = 0;
      min_value = raw_dat[i];
      if(min_value < (0.15*max_value)) beg_found = TRUE;
                              /* Definitely found end */
   }
   if(end_found == TRUE)
   {
      min_value = min(min_value,raw_dat[i]);
      if(min_value == raw_dat[i]) end_time = time; /* Found a better end */
      if(min_value < (0.15*max_value)) beg_found = TRUE;
                              /* Definitely found end */
   }
   if((beg_found==TRUE) && (end_found==TRUE))
   {
      time_arr[iii] = beg_time;
      time_arr[iii+1] = peak_time;
      time_arr[iii+2] = max_value;
      time_arr[iii+3] = end_time;
      if(peak_found == FALSE)
      {
       beg_time = time;
       beg_found = TRUE;
      }
      else
      {
       beg_found = FALSE;
       if(data_type == 'm') thershold = 0.02;
      }
      peak_found = FALSE;end_found = FALSE;
      iii+=4;
```

```
        if((data_type == 'm') && (max_value < (0.3*max_y))) iii -= 4;
        max_value=-5000;min_value=5000;
      }
      time += time_inv;
     }
    farfree((void *)raw_dat);
   }
  while(finish_flag==FALSE);
  if(peak_found == TRUE)
  {
   time_arr[iii] = beg_time;
   time_arr[iii+1] = peak_time;
   time_arr[iii+2] = max_value;
   if((beg_found == FALSE) && (end_found == TRUE))
     time_arr[iii+3] = end_time;
   else time_arr[iii+3] = stop_time;
   iii += 4;
   if((data_type == 'm') && (max_value < (0.3*max_y))) iii -= 4;
  }
  numwords[0] = iii/4;
  return(time_arr);
}

/************************************************************************
  The following strips raw data out of data file, skips data intervals,
  & returns data.

  Inputs:
   beg_time - beginning time
   end_time - ending time
   mult_num - word multiple number
   filename - complete file name (i.e. contains complete path & drive).
   data_type - (p)hoto or (m)ic
   skip_intv - skip data interval size

  Outputs:
   n - size of output buffer
   out_buf - processed output buffer

  Global:
   complete_file_name - filename to be processed (includes path & drive)
   start_time & stop_time - used in convert_data()
************************************************************************/
float *find_data(char filename[],char data_type[],float beg_time,
        float end_time,unsigned mult_num,unsigned *skip_int,unsigned long *n)
```

```
{
 float huge *convert_data(unsigned *,unsigned,char[],unsigned long *);
 double fract,intpart;
 float *out_buf,huge *raw_dat,fs;
 unsigned long buf_size,max_iii,beg_n=0,end_n=0,i=0,ii=0,iii=0;
 unsigned finish_flag,max_mult_num;

 strcpy(complete_file_name,filename);
 get_file_info(&max_mult_num,&numbytes);
 /* where: max_mult_num = number of words and/or sentences &  */
 /*        numbytes = window size used to record data */
 close(file_handle); /* "file_handle" is a global variable used by */
                     /* get_file_info to open files and leave open */
 start_time = 0;stop_time = numbytes/22.0/2500.0;
 if(data_type[0] == 'p') fs = 2500.0;
 else fs = 25000.0;
 beg_n=(unsigned long)floor(beg_time*fs/(float)skip_int[0])*skip_int[0];
 end_n=(unsigned long)ceil(end_time*fs/(float)skip_int[0])*skip_int[0];
 if(end_n > (unsigned long)(stop_time*fs))
  end_n = (unsigned long)floor(stop_time*fs/(float)skip_int[0])*skip_int[0];
 max_iii = (end_n-beg_n)/(unsigned long)skip_int[0] + 1L;
 if(max_iii > farcoreleft()/2/8)
 {
  max_iii = farcoreleft()/2/8;
  skip_int[0]  = (unsigned)((end_n-beg_n)/(max_iii - 1L));
 }
 if((out_buf = farcalloc(max_iii+1,sizeof(float))) == NULL)
 {
  printf("ERROR(1):  out_buf buffer allocation failed!\n");
  printf("Location:  iso_pho_env() routine in proc_spc.c.\n");
  perror("");exit(0);
 }
 finish_flag=TRUE;ii = 0;iii = 0;
 do                                 /* Find beg. & end of words */
 {
  raw_dat = convert_data(&finish_flag,mult_num,data_type,&buf_size);
  for (i=0;i<buf_size;i++)          /* Store points in out buffer */
  {                          /* & skip every "skip_int" one */
   fract = modf((double)(ii/skip_int[0]),&intpart);
   if((fract == 0) && (ii >= beg_n) && (ii <= end_n))
   {
      if(data_type[0] == 'p') raw_dat[i] += 2.5;
      out_buf[iii] = raw_dat[i];
      iii++;
      if(iii >= max_iii)
```

```
        {
         finish_flag = TRUE;
         break;
        }
    }
    ii++;
    if((ii > end_n) || (iii >= max_iii))
    {
        finish_flag = TRUE;
        break;
    }
   }
   farfree((void *)raw_dat);
  }
  while(finish_flag==FALSE);
  if(iii > max_iii) iii = max_iii;
  *n = iii;
  return((void *)out_buf);
 }

float calc_pitch(float arr[],unsigned long count,float freq_res)
{
 unsigned long i,start_i;
 float max_amp,freq,fo,amp,thershold;

 thershold = 100000;
 max_amp=-500;
 start_i = floor(80.0/freq_res);
 for(i=start_i;i<count;i++)    /* Finding 1st max */
 {
  freq = i*freq_res;
  amp = arr[i]*arr[i]*arr[i];
  max_amp = max(max_amp,amp);
  if(max_amp > thershold)
  {
   if(max_amp == amp) fo = freq;
   else break;
  }
 }
 return(fo);
}
```

"FFT.c" Source Program

```
/************************************************************************
 PROGRAM: 1 DIMENSIONAL FAST FOUTIER TRANSFORM PROGRAM
 ALGORITH: DECIMATION IN TIME, RADIX 2, INPLACE FAST FOURIER
TRANSFORM
 AUTHORS: COOLEY, LEWIS AND WELCH
 CODE: MATT DIERKING - USAF/FORIEGN TECHNOLOGY DIVISION
 VERSION: 5 MAR 85

************************************************************************

   THIS PROGRAM CALCULATES THE DISCRETE FOURIER TRANSFORM FOR
 A
 SEQUENCE OF DATA OF LENGTH  N, WHERE N IS EQUAL TO M**2.  THE
 CALCULATION IS DONE USING THE COOLEY AND TUKEY ALGORITHM FOR
 DECIMATION IN TIME, RADIX 2.


 PROGRAM INPUTS:
        A_R   FLOAT    THIS IS THE INPUT REAL DATA SEQUENCE FOR WHICH
                     THE FOURIER TRANSFORM IS DESIRED.
        A_I   FLOAT  THIS IS THE INPUT IMAGINARY DATA SEQUENCE FOR
WHICH
                     THE FOURIER TRANSFORM IS DESIRED.
        M     INTEGER  THE POWER OF TWO WHICH INDICATES THE NUMBER
                     OF SAMPLES IN THE SEQUENCE.
        INV   INTEGER  FLAG TO INDICATE WHETHER THE TRANSFORM OR
ITS
                     INVERSE IS TO BE CALCULATED.
                      0 - TRANSFORM
                      1 - INVERSE TRANSFORM

 PROGRAM OUTPUTS:
       A_R & A_I
           FLOAT    THESE ARE THE FOURIER TRANSFORMS OF THE INPUT
                    SEQUENCE.  THIS IS AN INPLACE ALGORITH, IE.
                    THE OUTPUT OF THE ALGORITHM REPLACES THE
                    INPUT DATA.

************************************************************************/

#include <dos.h>
#include <math.h>
```

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <graphics.h>

#define pi 3.14159265358979

typedef struct
          {
          float R;
          float I;
          } complex;

fft2(float huge *A_R,float huge *A_I,unsigned M,int INV)
{
 unsigned long I,IP,J,K,N,L,LE,LE1,NM1,NV2;
 complex U,U1,W,T;
 float   ANG;

 U.R = 0;U.I = 0;W.R = 0;W.I = 0;T.R = 0;T.I = 0;
 N   = (unsigned long) 1 << M;
 NV2 = N / 2;NM1 = N-1;J   = 1;

 for (I=1; I<=NM1; I++)
 {
  if (I < J)
  {
   T.R      = A_R[J-1];
   T.I      = A_I[J-1];
   A_R[J-1] = A_R[I-1];
   A_I[J-1] = A_I[I-1];
   A_R[I-1] = T.R;
   A_I[I-1] = T.I;
  }
  K = NV2;
  while (K < J)
  {
   J = J-K;
   K = K/2;
  }
  J=J+K;
 }
 for (L=1; L<=M ;L++)
 {
```

```
LE  = (unsigned long) 1 << L;
LE1 = LE/2;
U.R = 1.0;
U.I = 0.0;
ANG = pi/ (float) LE1;
W.R = cos(ANG);
W.I = sin(ANG);
if (INV != 0) W.I = -W.I;
for (J=1; J<=LE1; J++)
{
 for (I=J; I<=N; I+=LE)
 {
     IP  = I+LE1;
     T.R = (A_R[IP-(unsigned long)1] * U.R) - (A_I[IP-(unsigned long)1] * U.I);
     T.I = (A_R[IP-(unsigned long)1] * U.I) + (A_I[IP-(unsigned long)1] * U.R);
     A_R[IP-(unsigned long)1] = A_R[I-(unsigned long)1] - T.R;
     A_I[IP-(unsigned long)1] = A_I[I-(unsigned long)1] - T.I;
     A_R[I-(unsigned long)1] = A_R[I-(unsigned long)1] + T.R;
     A_I[I-(unsigned long)1] = A_I[I-(unsigned long)1] + T.I;
 }
 U1.R = (U.R * W.R) - (U.I * W.I);
 U1.I = (U.R * W.I) + (U.I * W.R);
 U.R  = U1.R;
 U.I  = U1.I;
}
if (INV != 0)
 for (I=1; I<=N; I++)
 {
     A_R[I-1] /= N;
     A_I[I-1] /= N;
 }
}
return;
}
```

"Conv_dat.c" Source Program

```
/***************************************************************************
Data convertion program that strips out merged photo data from
microphone data and returns either. The routine fills a buffer with
converted float data from a binary file.

I/p & O/p varibles:
  finished_flag:     If i/p - tells routine that this is a new run.
                     If o/p - tells calling routine whether this
                     routine is finished converting data.
  plot_num:          I/p only - Tells this routine which multiple word
                     number to convert (e.g., if plot_num = 3 then this
                     routine will convert the 3rd multiple of a
                     particular word.

  data_type:         I/p only - Tells this routine which data type
                     (mic or photo) to convert.
  store_buf_size:    O/p only - Tells calling routine size of converted
                     buffer.
Global varibles:
  complete_file_name:      (I/p only) Calling routine must provide this
                     (include drive and path). This routine will then
                     call "get_file_info" that uses this variable to open
                     a particular word file (e.g., "rawdat.000"). It also
                     leaves the file open and data pointer set 1st byte
                     to be converted.
  file_handle: (I/p only) Returned after calling "get_file_info".
                     This routine will used read from data file and
                     then close the file.
  start_time:        (I/p only) This routine uses this time to find
                     starting position of file left open after calling
                     "get_file_info". Since this is a global variable
                     used by other routines, the calling routine must
                     reset "start_time" to original value when finished.
  stop_time:         (O/p only) This routine uses this time to tell
                     calling routine ending position of word file.
                     Since this is a global variable used by other
                     routines, the calling routine must reset "start_time"
                     to original value when finished.

Program: conv_spc.c
Programmer: Patrick T. Marshall
Date: 2/25/91
```

```
 Organization:  WRDC/AAWP-2,
                WPAFB, OH 45433
 Phone:  (513) 255-2471

*************************************************************************/
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\conv_spc.h"

void get_file_info(unsigned *,unsigned long *);

float huge *convert_data(unsigned *finish_flag,unsigned plot_num,
                         char data_type[],unsigned long *store_buf_size)
{
 unsigned long rnd_off(float);
 float huge *store_buf;
 float max_t,fs,volts;
 unsigned exit_routine=0;
 unsigned int far *read_buf;
 unsigned tol_num_plots,max_par;
 unsigned long beg_byte,floor_val;
 unsigned long read_buf_size=0,max_int_count=0,store_count=0,int_count=0;
 unsigned long i,num_floats=0,start_mem_avail=0;
 static float start_t,stop_t

 start_mem_avail = farcoreleft();
                    /* Open file and leave open */
 get_file_info(&tol_num_plots,&numbytes);
 /* where: tol_num_runs = number of words and/or sentences &  */
 /*        numbytes = window size used to record data */
 if(plot_num > tol_num_plots)
 {
   printf("ERROR(1):  plot_num > tol_num_plots in conve_data routine\n");
   exit(1);
  }
  max_t = numbytes/22.0/2500.0;
  if (*finish_flag == 1)          /* New run */
  {
   if ((stop_time > max_t) || (stop_time < start_time))
    stop_t = max_t;
   else
    stop_t = stop_time;
   if ((start_time > max_t) || (start_time > stop_time))
    start_t = 0;
   else
    start_t = start_time; /* static var. controled by this routine */
  }
  else                             /* Continue old run */
  {
   start_t = stop_t; /* static var. controled by this routine */
   stop_t = stop_time;
  }
  tol_mem_avail = farcoreleft();
  if (*data_type == 'p')                  /* Photo data */
  {
   fs = 2500.0;
   beg_byte = rnd_off(2.0*fs*start_t*11);
  }
  else                             /* MIC data */
  {
   fs = 25000.0;
   beg_byte = rnd_off(2.0*(fs*start_t+1+floor(fs*start_t/10.0)));
  }
    /* Throw first 4 bytes & position pointer start byte */
 lseek(file_handle,beg_byte+(plot_num-1)*numbytes+6,SEEK_SET);
 int_count = floor((float)(beg_byte/2.0));
 num_floats = floor((fs*(stop_t - start_t)));
 if (num_floats*4 > tol_mem_avail)          /* Won't reach end */
 {
  *finish_flag = 0;
  store_buf_size[0] = floor(0.98*tol_mem_avail/4.0);       /* Stop when buffer used >=
98% */
 }                                          /* of available memory */
 else                                /* Will reach end */
 {
  *finish_flag = 1;
  store_buf_size[0] = num_floats;   /* Stop when buffer used = number floats */
```

```
}
if ((store_buf = farcalloc(store_buf_size[0],sizeof(float))) == NULL)
{
 printf("ERROR(3):  storage buffer allocation failed in convert_data routine.\n");
 perror("");
 exit(1);
}
tol_mem_avail = farcoreleft();
if (tol_mem_avail > 64000)          /* Check for read buffer size limitation */
 read_buf_size = 32000;
else
 read_buf_size = floor(tol_mem_avail/2.0);;
if (*data_type == 'p')              /* Photo data */
 max_int_count = 11*store_buf_size[0];
else                                /* MIC data */
 max_int_count = 1+floor((float)store_buf_size[0]/10.0)+store_buf_size[0];
max_int_count += int_count;
if (read_buf_size > max_int_count) read_buf_size = max_int_count;
if ((read_buf = farcalloc(read_buf_size,sizeof(int))) == NULL)
{
 printf("ERROR(2):  read buffer allocation failed in convert_data routine.\n");
 perror("");
 exit(1);
}
do
{
 if ((read(file_handle,read_buf,read_buf_size*
            sizeof(int))) == -1)
 {
  printf("ERROR(4):  read failed in convert_data routine.\n");
  perror("");
  exit(1);
 }
 for (i=0;i<read_buf_size;i++)
 {
  if (read_buf[i] > 4096)
  {
     printf("ERROR(5) in convert_data:  Byte Error - read_buf > 4096!\n");
     exit(1);
  }
  volts = 5.0/4096.0*(float)(read_buf[i])-2.5;
  if ((volts > 2.5) || (volts < -2.5))
  {
     printf("ERROR(6):  Volts = %f \n",volts);
     printf("Location: convert_data() routine.\n");
```

```
          getch();
          exit(1);
        }
        if ((fmod((float)int_count,11) == 0) && (*data_type == 'p'))
        {   /* Photo data */
            store_buf[store_count] = volts;
            store_count++;
        }
        if ((fmod((float)int_count,11) != 0) && (*data_type == 'm'))
        {           /* MIC data */
          store_buf[store_count] = volts;
          store_count++;
        }
        int_count++;
        if ((store_count >= store_buf_size[0]) || (int_count>=max_int_count))
        {
          exit_routine = TRUE;
          break;                          /* Exit for loop */
        }
      }
      if (exit_routine == TRUE) break;
      if (int_count+read_buf_size>max_int_count)
       read_buf_size = max_int_count-int_count;
    }
    while (exit_routine == FALSE);
    stop_t = start_t + (float)store_count/fs;
    close(file_handle); /* "file_handle" is a global variable used by */
                        /* get_file_info to open files and leave open */
    farfree(read_buf);
    return(store_buf);
   }

   /* Round up-or-down routine */
   unsigned long rnd_off(float number)
   {
    double frac,intpart;
    unsigned long rnd_off_num;

    frac = modf(number,&intpart);
    if (frac >= .5) rnd_off_num = floor(number)+1;   /* Round up */
    else rnd_off_num = floor(number);                /* Round down */
    return rnd_off_num;
   }
```

"Warp.c" Source Program

```
/***************************************************************************
 Time warping program.

 Program:  warp.c
 Programmer:  Patrick T. Marshall
 Date:  11/19/91
 Organization:  WRDC/AAWP-2,
                WPAFB, OH 45433
 Phone:  (513) 255-2471

***************************************************************************/
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\warp.h"

/***************************************************************************
 The following is a Time-Warping routine.  Implements DP time-
 warping procedure to align two patterns, a & b.  Returns
 resultant min. cost & the warping function (here called "map").
 Stages in DP process are samples of pattern "a" (the test pattern).
 Reference:
  Parsons, Thomas W., "Voice and Speech Processing," pages 297-303
  and Appendix B, pages 379-382. New York: McGraw Hill Book Company,
  1987.
 Language:
  Fortran-77 converted to Borland's Turbo C by Patrick T. Marshall,
  Mar., 1991.
 Variables:
  I/P's:
   a:  1st pattern (test pattern along x-axis)
   m:  number of points in "a" (related to "i" no. of col.'s)
   b:  2nd pattern (reference (template) pattern along y-axis)
   n:  number of points in "b" (related to "j" no. of rows)
  O/P's:
   tcost:      cost of optimum path
   map:        encoded optimum path (size = m+n-1)
   k:  length of optimum path
  Internal:
   cost:       2-D array of recent accrued costs (n rows by 3 cols)
   pred:       2-D array of predecessor points (n rows by m cols)
   steps:   1-D array of +/- integers for slope constraint. "+"
               are for vertical steps and "-" are for horizontal
               steps.
 Constaints and Limitations:
  1. Maximun tempate length is limited by size of cost & pred
     arrays. This is currently set to 100 which should allow
     a maximum word length of 1.67 sec (1.67 sec = 100/60 Hz
     for positive peaks only).
  2. Accrued costs are retained for only two most recent rows of
     points. At the end of a row, costs for row 2 (current row)
     replace costs for row 1 (previous row) which are no longer
     needed.
  3. Paths & predecessor-point coordinates are encoded and decoded
     into a single 16-bit unsigned integer by means of functions,
     "epfunc() & dpfunc()" respectively.
  4. The parallelogram method is used to define a "window_width path".
     This is fine provided that "m & n" are approximately equal.
     If m = 2n or n = 2m, the prallelogram collapses into a straight
     line (i.e., no warping is accomplished).
***********************************************************************/

extern int window_width,max_step;

unsigned *warp(float a[],unsigned cols,float b[],unsigned rows,float *tcost,
            unsigned *k)
{
 void set_up_plt(char[],double,double,double,double,char[]);
 void plot(float huge *,float,float,unsigned long,int);
 void plot2(float huge*,float huge*,unsigned,int);
 void print_plot(void);
 void erase_plot(void);
 void init_plot();
 float cfunc(float,float);
 int epfunc(unsigned ,unsigned );
```

```
   void dpfunc(unsigned,unsigned *,unsigned *);
   unsigned nint(float);
   float c,c1,c2,c3,slope,offset,huge *cost,HUGE = 1.0e37;
   unsigned lim1,lim2,huge *map,huge *step_cnt;
   unsigned x=0,y=0,i,j,ii,jj;
   unsigned long huge *pred,kk;
   float buf_x[2],buf_y[2];
   char file[31]="",pltpos[3],print_flag;

                          /* Allocate rows by cols.'s for pred[i][j] */
   if ((pred = farcalloc((rows+1)*(cols+1),sizeof(unsigned long))) == NULL)
   {
    printf("ERROR(1):  pred buffer allocation failed!\n");
    printf("Location:  warp() routine.\n");
    printf("rows*cols = %d \n",cols*rows);
    perror("");
    getch();
    exit(1);
   }
   if ((step_cnt = farcalloc(rows+1,sizeof(unsigned))) == NULL)
   {
    printf("ERROR(2):  steps buffer allocation failed!\n");
    printf("Location:  warp() routine.\n");
    perror("");
    getch();
    exit(1);
   }             /* Allocate rows+1 rows by 2 cols.'s for cost[i][j] */
   if ((cost = farcalloc((rows+1)*2,sizeof(float))) == NULL)
   {
    printf("ERROR(4):  cost buffer allocation failed!\n");
    printf("Location:  warp() routine.\n");
    printf("rows*3 = %d \n",3*rows);
    perror("");
    getch();
    exit(1);
   }
 /*
 init_plot();
 strcpy(&file[0],"c:\\borlandc\\thesis\\map.dat");
 set_up_plt(file,cols,175,rows,175,"c");
 */
  slope = (float)(rows-1)/(float)(cols-1);
  offset = (float)(rows - slope*cols);
  for (i=1;i<=cols;i++)          /* Loop through "a" */
  {                              /* Set max. allowable "Window Width" */
```

```
lim1 = max(1,nint(slope*i+offset-window_width));
lim2 = min(rows,nint(slope*i+offset+window_width));
for (j=lim1;j<=lim2;j++)   /* Loop through "b" */
{
 c = cfunc(a[i],b[j]);          /* Cost for this point */
                                /* Cost for path to this point is ...*/
 if ((i == 1) && (j == 1)) /* No predecessors */
 {
    cost[1+j*2] = c;
    pred[i+j*cols] = epfunc(1,1);
 }
 else                           /* Must consider 3 possible predecessors */
 {
    c1 = HUGE;
    if (pred[(i-1)+j*cols] > 0)
     c1 = cost[0+j*2] + c;         /* Horizontal cost */
    c2 = HUGE;
    if (pred[(i-1)+(j-1)*cols] > 0)
     c2 = cost[0+(j-1)*2] + c;     /* Diagonal cost */
    c3 = HUGE;
    if (pred[i+(j-1)*cols] > 0)
     c3 = cost[1+(j-1)*2] + c;     /* Vertical cost */
    if ((step_cnt[j] > 0) && (c2 < HUGE))
    {
     c1 = HUGE;
     c3 = HUGE;
    }
    if (step_cnt[j] > 0) step_cnt[j]--;
                                /* Find cheapest cost */
    if ((c1 >= HUGE) && (c2 >= HUGE) && (c3 >= HUGE))
     pred[i+j*cols] = 0;
    else if ((c1 <= c2) && (c1 <= c3))
    {
     cost[1+j*2] = (float)c1;
     pred[i+j*cols] = epfunc(i-1,j);
    }
    else if ((c2 <= c1) && (c2 <= c3))
    {
     cost[1+j*2] = (float)c2;
     pred[i+j*cols] = epfunc(i-1,j-1);
    }
    else if ((c3 <= c1) && (c3 <= c2))
    {
     cost[1+j*2] = (float)c3;
     pred[i+j*cols] = epfunc(i,j-1);
```

```
          }
        }
   /*
   kk = pred[i+j*cols];
   if((kk != 0) && (i !=1) && (j !=1))
   {
    dpfunc(kk,&x,&y);
    buf_x[0] = x;buf_y[0] = y;
    buf_x[1] = i;buf_y[1] = j;
    plot2(buf_x,buf_y,2,2);
   }
   */
                   /* Check for continuous horizontal runs */
      if((i>max_step) && (step_cnt[j] == 0))
          for(ii=i;ii>i-max_step;ii--)
          {
           kk = pred[ii+j*cols];         /* Find predecessor */
           if(kk==0) break;
           dpfunc(kk,&x,&y);             /* Decode kk */
           if(y != j) break;
           if(x == i-max_step) step_cnt[j] = max_step;
          }              /* Check for continuous vertical runs */
      if((j>max_step) && (step_cnt[j] == 0))
          for(jj=j;jj>j-max_step;jj--)
          {
           kk = pred[i+jj*cols];         /* Find predecessor */
           if(kk==0) break;
           dpfunc(kk,&x,&y);             /* Decode kk */
           if(x != i) break;
           if(y == j-max_step) step_cnt[j+1] = max_step;
          }
     }                                /* End of "b" loop */
    for (j=lim1;j<=lim2;j++)          /* Shift costs down */
      cost[0+j*2] = cost[1+j*2];
    }                                 /* End of "a" loop */
    *tcost = cost[1+rows*2];                /* Note total cost */
   /*
   printf("cost = %f\n",*tcost);
   if (getch() == 'p') print_plot();
   erase_plot();
   */
    farfree((void *)cost);
    if ((map = farcalloc(cols+rows,sizeof(unsigned))) == NULL)
    {
     printf("ERROR(5):  map buffer allocation failed!\n");
   printf("Location:  warp() routine.\n");
   perror("");
   getch();
   exit(1);
  }
  kk = epfunc(cols,rows);                  /* Work backward from final point */
  for (ii=cols+rows-1;ii>=1;ii--)
  {
   map[ii] = kk;
   dpfunc(kk,&i,&j);           /* Decode kk */
   if ((i == 1) && (j == 1)) break;
   if ((i <= 0) || (j <= 0))
   {
    printf("ERROR(6):  decode kk failed!\n");
    printf("Location:  warp() routine.\n");
    getch();
    exit(1);
   }
   if ((i > cols) || (j > rows))
   {
    printf("ERROR(7):  decode kk failed!\n");
    printf("Location:  warp() routine.\n");
    getch();
    exit(1);
   }
   kk = pred[i+j*cols];                    /* Find next predecessor */
  }
  *k = cols + rows - ii;ii--;
  if (ii > 0)              /* Shift map down to start of array */
   for (i=1;i<=*k;i++)
     map[i] = map[i + ii];
  farfree((void *)step_cnt);
  farfree((void *)pred);
  return((void*)map);
}

/* Cost function used by warp() */
float cfunc(float x,float y)
{
 float c;

 c = pow(x-y,2);
 return(c);
}
```

```
/* Path encoding function used by warp() */
int epfunc(unsigned i,unsigned j)
{
 unsigned p=0;

 p = 256 * i + j;
 return(p);
}

/* Path decoding function used by warp() and others */
void dpfunc(unsigned kk,unsigned *i,unsigned *j)
{
 *i = floor(kk/256);
 *j = (unsigned)fmod(kk,256);
 return;
}

/* Nearest integer function used by warp() */
unsigned nint (float x)
{
 float p=0;
 unsigned q;

 p = floor(x)+0.5;
 if(p<0) q = 0;
 else if(p>=x) q = floor(x);
 else q = ceil(x);
 return(q);
}

/************************************************************************
 The following is a Linear Time-Warping routine. Implements time-
 warping to align two patterns, a & b, with respect to their peaks.
 Returns warped function (here called "warped_buf").

 Variables:
  I/P's:
   a:  Pattern to be warped
   m:  number of points in "a"
   b:  Reference pattern (template)
   n:  number of points in "b"
  O/P's:
   warped_buf:  resultant warped a buffer
************************************************************************/
float *linear_warp(float a[],unsigned m,float b[],unsigned n)
```

"Acq_dat.c" Source Program

```
/*---------------------------------------------------------------

Program contains aquisition speech functions used by main.

Program:  aqc_spc.c
Programmer:  Patrick T. Marshall
Date:  2/25/91
Organization:  WRDC/AAWP-2,
               WPAFB, OH 45433
Phone:  (513) 255-2471

Notes:

 1. There are 12 isolated words numbered "0" through "11".  In addition,
 there are 7 continuous words numbered "12" through "18".  If the external
 char variable "op_mode[0]" defined in acq_spc.h (originally from
 data_acquisition() routine in speech4.c) is equal to "i" then work
 in the isolated mode.  Otherwise assume continuous mode.
-----------------------------------------------------------------------*/
#pragma check_stack(off)
#include <bios.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <dir.h>
#include <string.h>
#include "c:\borlandc\thesis\logical.h"
#include "c:\borlandc\thesis\acq_spc.h"

/* Function prototypes */
int alloc_dma_buf(void);        /* Allocate dma buffers */
void intr_setup(void);          /* Set up interrupt operation */
void dma_setup(void);                  /* Set up dma operation */
void dma_finish(void);          /* Called via atexit() mechanism */
void interrupt far dma_isr(void);
void start_dma(char far *,unsigned); /* Start a dma operation */
void init_brd(void);            /* Initialize A/D board */
void on_brd(void);              /* Turn A/D board on */
void acquire_data(void);
void wait_message (int, int, char*, char*);
void message (int, int, char*, char*);
void clear_message (void);
void build_path (char*, char*, char*, int);
void restore_window(void);

void acquire_data()
{
 unsigned i,offset,word_scram_offset;
 unsigned long count,num_bytes=0,byte_count=0;
long mem;
 int next_count,prev_count,start_count,run_cnt;
 int start_word,stop_word;
 char filename[81];
 struct dfree disk;
 FILE   *fileptr;          /* File pointer for word file "word.lst" */

 numbytes = (unsigned long)(2.0*(25000+2500)*ts);
                            /* Load the word array & create files */
 if(op_mode[0] == 'I')
 {
  start_word = 0;stop_word = num_words;start_count = 1;
  strcpy(filename,"c:\\borlandc\\thesis\\isoword.lst");
 }
 else
 {
  start_word = 12;stop_word = 12 + num_words;start_count = 13;
  strcpy(filename,"c:\\borlandc\\thesis\\contword lst");
 }
 if ((fileptr = fopen(filename,"r"))  == NULL)
 {
  wait_message(0,7,"WARNING:  fopen of word.lst file failed! ",
  "Location:  aquire_data() routine in ACQ_SPC.C");
 }
 for (count=start_word;count<stop_word;count++)
 {
  fgets(word_buffer[count],40,fileptr);
  build_path(complete_file_name,drive,path,count);
  if ((file_handle = open(complete_file_name,
        O_WRONLY|O_CREAT|O_TRUNC|O_BINARY|O_APPEND,S_IREAD|S_IWRI
TE)) == -1)
   {
    message(0,15,"ERROR(1): raw data file open failed! \n",
         "Location: aquire_data() routine in ACQ_SPC.C");
   }
  if (write(file_handle,&num_runs,sizeof(int)) == -1)
        wait_message(0,7,"ERROR(2): write buffer failed!",
              "Location: aquire_data() routine in ACQ_SPC.C");
  if (write(file_handle,&numbytes,sizeof(unsigned long)) == -1)
        wait_message(0,7,"ERROR(3): write buffer failed!",
              "Location: aquire_data() routine in ACQ_SPC.C");
  close(file_handle);
 }
 prev_count = 0;
 if (fmod(num_words,2) == 0) word_scram_offset = 1;
 else word_scram_offset = 0;
 for (run_cnt=0;run_cnt<num_runs;run_cnt++)
 {
  next_count = start_count-1;
  for (count=start_word;count<stop_word;count++)
  {
   getdfree(0,&disk);
   mem=farcoreleft();
   message(0,7,"Hit any key!","");
   getch();restore_window();
   build_path(complete_file_name,drive,path,next_count);
   if ((file_handle = open(complete_file_name,O_WRONLY|O_BINARY|O_APPEND,
                                         S_IREAD|S_IWRITE)) == -1)
   {
      message(0,15,"ERROR(4): file open failed!",
      "Location: aquire_data() routine in ACQ_SPC.C");
   }
   textcolor (LIGHTRED);
   prev_count = next_count;
   message(0,10,word_buffer[next_count], "");
   if((*op_mode == 'i') || (*op_mode == 'I'))
   {  /* The following is for the isolated word scambler. */
      next_count = next_count + start_count;
      if (next_count > num_words-1)
        next_count = next_count - num_words - word_scram_offset;
      if ((start_count == num_words) && (next_count == prev_count))
        next_count = next_count - 1;
   }
   else
```

```
    next_count++;
 textcolor (LIGHTGRAY);
 init_brd();                    /* Initialize A/D board */
                                           /* Allocate buffers only once */
 alloc_dma_buf();
 if ((run_cnt == 0) && (count == start_word))  /* Only do this once */
 for (i=0;i<numbuffers;i++)              /* to insure will have exact */
    num_bytes += dma_buffers[i].s;  /* same window byte size/transfer */
 buf_index = 1;
 if(numbuffers > 1)
 {
    dma_chan = 3;
    dma_setup();                          /* Set chan 3 up for DMA operations */
    start_dma(dma_buffers[1].p,dma_buffers[1].s);  /* Start up data acq */
 }
 intr_setup();                            /* Set up DMA IRQ3 interrupt */
 dma_chan = 1;
 dma_setup();                             /* Set chan 1 up for DMA operations */
 start_dma(dma_buffers[0].p,dma_buffers[0].s);  /* Start up data acq */
 on_brd();
 delay(200);          /* Pause for 200 milliseconds */
 while(TRUE)
 {
    if (irq_flag)
    {
     buf_count++;
     if (irq_flag == 2) break;
     irq_flag = 0;
    }
 }
 for (i=0;i<numbuffers;i++)
 {
    byte_count += (unsigned long) dma_buffers[i].s;
    if (numbytes != num_bytes)
    {
     dma_finish();
     wait_message(0,7,"ERROR(5): byte window size changed between transfers",
       "Location:  aquire_data() routine in ACQ_SPC.C");
     return;
    }                         /* Throw out first 4 MIC bytes */
    if (i == 0) offset = 4;
    else offset = 0;
    if (write(file_handle,dma_buffers[i].p+offset,dma_buffers[i].s-offset) == -1)
    {
     dma_finish();
```

```
        wait_message(0,7,"ERROR(6): write buffer failed!",
        "Location: aquire_data() routine in ACQ_SPC.C");
        return;
       }
    }
    numbytes = (unsigned long)(2.0*(25000+2500)*ts);
    byte_count = 0;irq_flag = 0;
    dma_finish();
    restore_window();
    close(file_handle);
   } /* End of inner loop */
   start_count++;
   if (start_count > num_words) /* Have to reset counter*/
   {
    if((*op_mode == 'i') || (*op_mode == 'I')) start_count = 1;
    else start_count = 13;
   }
  } /* End of outer loop */
  /* Since "alloca_dma_buf()" modifies numbytes, have to rewrite data at */
  /* beginning of file. */
  num_bytes -= 4; /* Throw out first 4 MIC bytes */
  for (count=start_word;count<stop_word;count++)
  {
   build_path(complete_file_name,drive,path,count);
   if ((file_handle = open(complete_file_name,O_WRONLY|O_BINARY,
                                   S_IREAD|S_IWRITE)) == -1)
   {
    message(0,15,"ERROR(7): file open failed! \n",
    "Location: aquire_data() routine in ACQ_SPC.C");
    exit(0);
   }
   if (write(file_handle,&num_runs,sizeof(int)) == -1)
          wait_message(0,7,"ERROR(8): write buffer failed!",
              "Location: aquire_data() routine in ACQ_SPC.C");
   if (write(file_handle,&num_bytes,sizeof(unsigned long)) == -1)
          wait_message(0,7,"ERROR(9): write buffer failed!",
              "Location: aquire_data() routine in ACQ_SPC.C");
   close(file_handle);
  }
  fclose(fileptr);
  return;
 }
```

Bibliography

1. Petajan, Eric D. "Automatic Lipreading to Enhance Speech Recognition," IEEE Global Telecommunications Conference. 265-272. New York: IEEE Press, November 1984.

2. Alex Pentland and Kenji Mase "Lip reading: Automatic Visual Recognition of Spoken Words," M.I.T. Media Lab Vision Science Technical Report 117:1-9 (January 1989).

3. Kabrisky, Matthew. Thesis advisory meetings and telephone conversations. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1990 - August 1992.

4. Pyroelectric Infrared Sensor IRA Series. Product Catalog S01E-5. muRata Erie North America, Smyrna, GA, 1988.

5. Amperex Electronic Corporation Technical Publication 163, Slatersville Division, Smithfield, RI, undated.

6. Silicon Photodiodes Optoelectronics Data Book. EG&G Vactec Optoelectronics, St. Louis MO, 1990.

7. Photosensor Product Catalog. Advanced Optoelectronics, City of Industry CA, 1989.

8. Saito, Shuzo and Nakata, Kazuo. Fundamentals of Speech Signal Processing, Academic Press, 1985.

9. General Purpose Detectors Technical Data Sheet. Silicon Detector Corporation, Camarillo CA, undated.

10. Parsons, Thomas W. Voice and Speech Processing. McGraw-Hill Series in Electrical Engineering, 1987.

11. Bevington, Philip R. Data Reduction and Error Analysis for the Physical Sciences. McGraw-Hill Book Company, 1969.

12. Rabiner, L. R. and Schafer, R.W. Digital Processing of Speech Signal, Prentice-Hall, 1978.

13. Eggebrecht, Lewis C. Interfacing to the IBM Personal Computer, Howard W. Sams and Company, 1990.

14. MAX167ACNG A/D Specifications Sheet. MAXIM, Sunnyvale CA, 1990.

15. Nolan, Tom. "Real-Time Data Acquisition Using DMA," Dr. Dobb's journal, pages 28-37,94-96 (January 1990).

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE Sep 92	3. REPORT TYPE AND DATES COVERED Final Jun 90 - Jan 93

4. TITLE AND SUBTITLE Speech Recognition Using Visible and Infrared Detectors	5. FUNDING NUMBERS
6. AUTHOR(S) Patrick T. Marshall	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) AFIT/ENG, BLdg 640 2950 P St. WPAFB, OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GE/ENG/93M-01
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release: Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A system has been developed that tracks lip motion using infrared (IR) or visible detectors. The purpose of this study was to determine if the additional information obtained from the IR or visible detectors can be used to increase the recognition rate of audio Automatic Speech Recognition (ASR) systems. To accomplish this goal, several hardware analog prototypes had to be designed, built and tested. Different detectors (IR and visible) and modes of operation (active and passive) were tried before a reliable and useful signal was found. An analog-to-digital (A/D) board was then designed and built that digitized both the microphone and photo signals. Software algorithms, executed from a desktop PC, were used to interface with the A/D board, process the digitized data, and perform certain optical and audio ASR experiments. The results showed that isolated ASR audio recognition rates increased after using additional information gained from the photo speech signals. However, the results for the continuous case were inconclusive since not all of the available photo information was utilized to perform ASR experiments.

14. SUBJECT TERMS Speech recognition, IR, Visible, Detectors, Audio, DTW, Photo, Sensors, ASR, A/D	15. NUMBER OF PAGES 207
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102